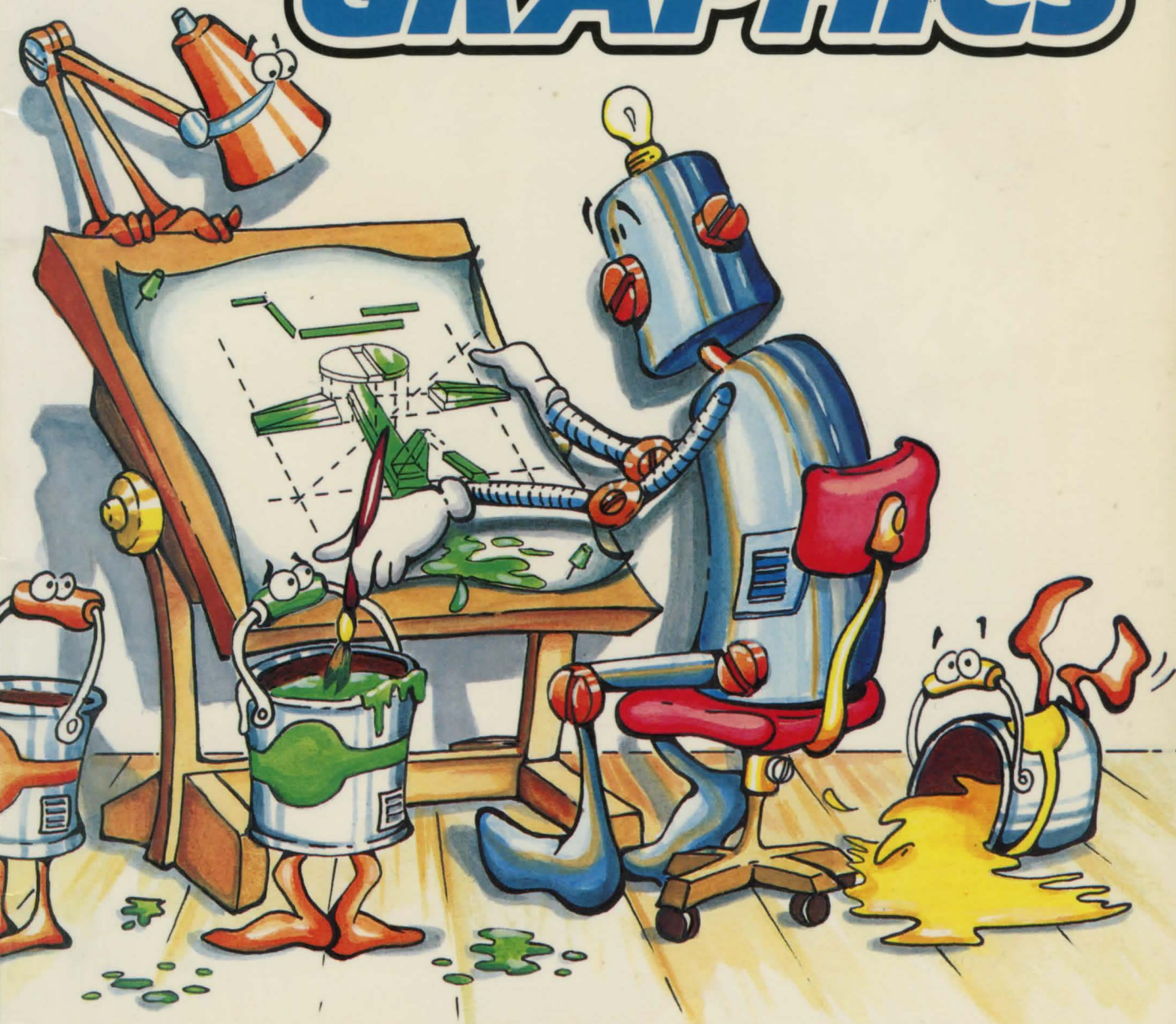
GETTING AHEAD

APPLE

APPLE GRAPHICS



HAYES

APPLE™ GRAPHICS

by Lynne Zucker and Greg Wilson

Editor: Carol Batchelor, B.A., M.A.
Illustration: Margaret Horvath
Tim O'Halloran
Design: Margaret Horvath
Pat Strachan

Hayes Publishing Ltd., 3312 Mainway, Burlington, Ontario. L7M 1A7

NOTE

The publisher and authors have made every effort to assure that the computer programs are accurate and complete. However, this publication is prepared for general readership, and neither the publisher nor the authors have any knowledge about or ability to control any third party's use of the programs and programming information. There is no warranty or representation by either the publisher or the authors that the programs or programming information in this book will enable the reader or user to achieve any particular result.

ISBN 0-88625-047-1

CONTENTS

About This Book 4
How to Use The Apple 6
How Graphics Works 8
Shape Tables 10
The Shape Table Program 12
Peek At A Poke 14
Color, Rotation and Scale 15
Moving Right Along 17
Smoother Sailing 18
Turning Corners 19
Making Sounds 20
Getting Control 22
How We Got Control 24
Games Paddles 28
Looking Back 29
The Game So Far 30
Shoot 'Em Up Apple 32
Ready Aim 34
Apple Sound Explosion 36
Turrets Sighted 37
Take Cover 39
And The Winner Is 41
Getting Even Better 42
Other Games You Can Make 43
Maze Games, Space Games 44
Bat And Ball Games 45
Glossary 46

ABOUT THIS BOOK

This book will show you, by example, how to write arcade style games for your Apple computer using the Applesoft BASIC language.

Computer Fundamentals

Even if you don't already know BASIC, this book will give you all you need to know to create your very own games. First it reviews the Apple computer fundamentals: how to start up the computer, load in diskettes, and run and save your programs.

Shapes, Sound And Color

Then "Apple Graphics Games" will tell you how to draw shapes on the screen, how to make them move and how to add color and sound to your programs. Games paddles and joysticks can make the games you write even more realistic and exciting. You'll find out how to use them too.

Programming

You will learn many game programming techniques as you type in the game called "Appleflight" that is developed in this book.

To use this book you will need an Apple microcomputer with 48K of memory, a disk drive and a display screen.

The glossary at the back will give you the meaning of some of the more common computer terms.

Bug Out!!
Debugging Your Program

No one is a perfect typist. Chances are good that you'll make at least one mistake when entering the program lines. You'll discover this (and any other bugs) when you enter RUN and nothing happens. When this occurs you should see an "error message" on the screen – or you may have to type TEXT first, and then the error message will be displayed. For example:

? SYNTAX ERROR 4125

This means the computer cannot understand what line 4125 is telling it to do. To check this out you type:

LIST 4125

and line 4125 will appear on the screen. Now check what you have with the correct line in the book. If you have made a mistake, simply re-type the line (correctly) and press RETURN. Then save Appleflight again and run the program.

You can list your whole program at any time by typing:
LIST (RETURN)
The screen scrolls quickly so use CTRL and S to stop it (and start it again). You can check your entire program line by line this way.

****Be sure you type LINE NUMBERS correctly, otherwise it will be that much harder to find a bug.**

*****Be especially careful that you don't confuse the CAPITAL LETTER I and the NUMBER 1 (one) while typing!**


APPLE FLIGHT

HOW TO USE THE APPLE

This is a very quick review of how to turn on the Apple, and how to save and run your programs. The Applesoft Tutorial and the Basic Programming Reference Manuals that came with your computer contain more details if you need them.

Starting Up

On most Apple microcomputers you must insert a diskette called "System Master" into the disk drive before you can turn on the machine. To do this, open the flap on the front of the disk drive labelled "Drive 1". (If you only have one disk drive, that's the one to use!) Gently insert the disk with the label facing up. Close the flap and you are ready to "boot" or turn on the computer.

To turn on the computer use the switch at the back of the computer on the left-hand side. The computer is on when the button on the keyboard labelled "POWER" is lit up. On some older computers, turning on the computer isn't enough to boot the system. If you turn on the computer and the disk drive doesn't begin to whirr, then type this:
PR#6
and press the RETURN key.

The disk drive will begin to whirr (it's reading information from the disk), and it is done when the red light on the front of the disk drive goes out.

Remember, you have to turn on your TV or display monitor too.

Running Programs

After you have booted the computer using the System Master diskette, type the following lines. Remember after you finish typing each line press the RETURN key to move to the next line.

```
NEW
10  FOR I=1 TO 24
20  PRINT "LET'S WRITE GRAPHICS GAMES! ! ! ! !"
30  NEXT I
40  END
RUN
```

The command "NEW" tells the computer that we wish to start a new program and to erase any old program still in the computer's memory.

This example program prints "LET'S WRITE GRAPHICS GAMES ! ! ! ! !" all the way from the top to the bottom of your screen. This programmed repetition is called a "loop." The actual program is the four lines which begin with numbers. Program lines always begin with

a number. You can look at the program you have in memory at any time by typing:

LIST

Of course, the command "RUN" is the signal to the computer to go ahead and do what the program instructions tell it to do.

Saving Programs

It would be a good idea for you to get a blank diskette to store the programs from this book. Before a blank diskette can be used it must be "initialized." (Make sure the disk is blank or that the programs on it are no longer needed. The INIT command will erase the whole disk.) When you are ready to initialize a new diskette, take out the System Master and put the new disk into the disk drive. Now type:

INIT GAMES DISK, D1

The disk drive will turn for a long time. When the prompt ( ]) appears, type:

CATALOG

Now you have a blank diskette entitled "Games Disk" for storing the programs from this book. You can use the "CATALOG" command at any time to see what programs have been stored on a disk.

To save a program you must give it a name. Let's call the program on Page 6, "APPLE." To save the program, type:

SAVE APPLE

When the disk drive has finished running and the flashing prompt is back on the screen, type:

CATALOG

again and you will see that "APPLE" is now listed on the disk.

To show how to retrieve a program from a diskette we will clear memory by typing:

NEW

Now if you type:

LIST

there is no program in memory. Type this:

LOAD APPLE

When the prompt appears, type:

LIST

and you will see that you did get the program from the disk. Another way to retrieve a program is to give the command to load and run the program.

Type in this:

NEW

RUN APPLE

This time the program "APPLE" was both loaded into memory and RUN in one command.

HOW GRAPHICS WORKS

Type this:

```
NEW
10 GR
20 COLOR=3
30 PLOT 20,20
40 END
RUN
```

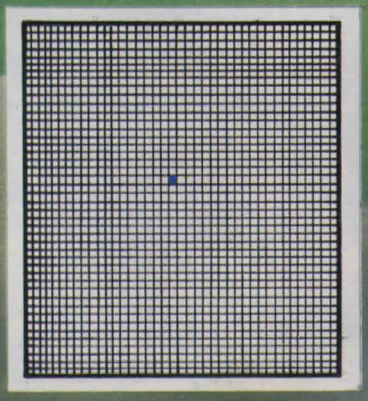
Type in this program:

```
NEW
10 HOME
20 VTAB10
30 INPUT"TYPE YOUR NAME:";NAME$
40 PRINT:PRINT
50 PRINT"HELLO  ";NAME$;"!!!!"
60 END
RUN
```

If you look closely at the screen, you will notice that the letters are made up of little dots. These are called pixels. When the dots are used to make letters on the screen the computer is in something called "text mode." The same pixels can be used to make up shapes and pictures, but not in text mode. To draw pictures the computer must first be in a "graphics mode" and then we have to give it a special series of drawing commands.

You can draw pictures using blocks of pixels or individual pixels. For blocks you must first put the computer in "low resolution graphics mode." Individual pixels are used with the computer in "high resolution graphics mode." Let's talk about "low res" graphics first and then move on to "high res."

Type in the next program:



a pixel

This shows you one low res block of pixels. In low resolution graphics the screen is divided into 40 blocks across and 40 blocks down, which leaves 4 lines at the bottom of the screen for letters. PLOT 20,20 tells the computer to light up the block located at a point 20 blocks across and 20 blocks down. You can make pictures by turning on different groups of blocks in different locations. Try changing line 30 to light up different blocks, but don't make the numbers bigger than 40!

To get back into text mode to type in another program or to change the old one you simply type:

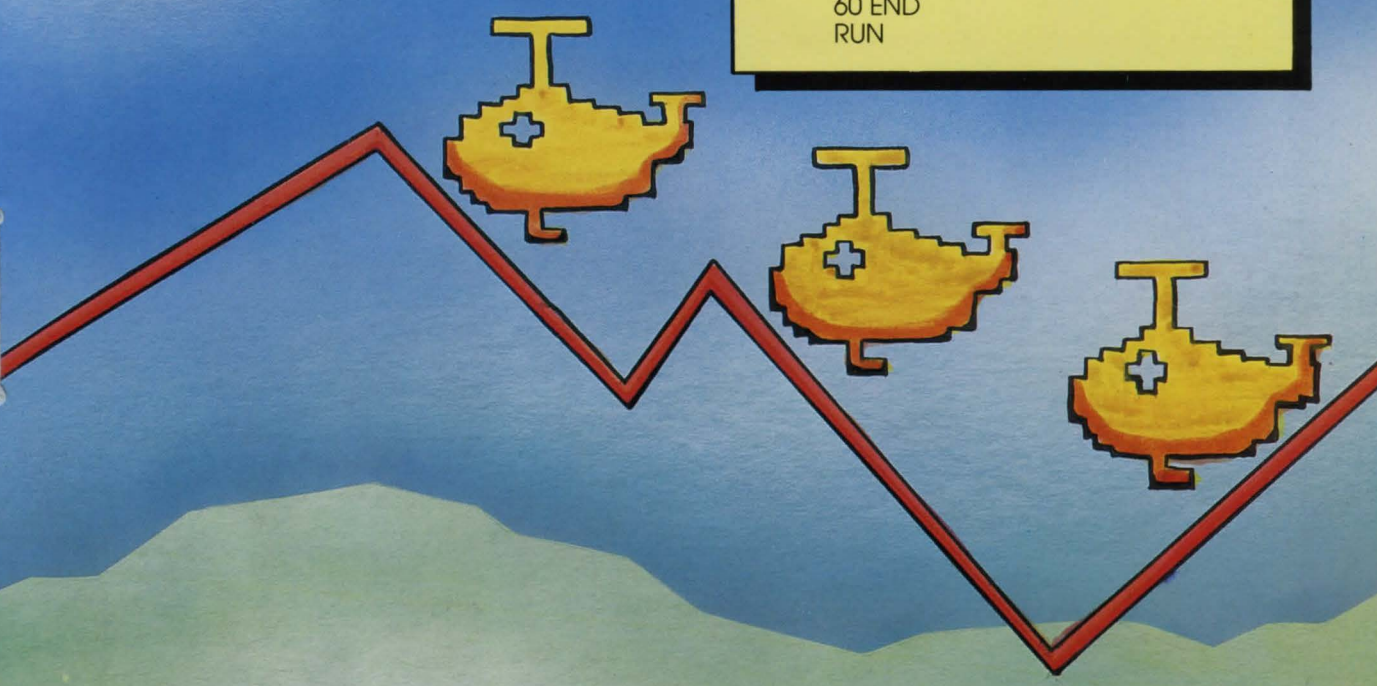
TEXT

Let's do a program similar to the last one, except that this time we'll tell the computer to go into high res mode. Type this:

```
NEW
10 HGR
20 HCOLOR=3
30 HPLOT 100,100
40 END
RUN
```

We can use the HPLOT command to make a series of lines into shapes. Type in this program and you will see one of the shapes used in our game "Appleflight."

```
NEW
10 HGR
20 HPLOT 100,100 TO 105,100
30 HPLOT 105,100 TO 110,105
40 HPLOT 110,105 TO 120,105
50 HPLOT 113,104 TO 122,104
60 END
RUN
```



This time only one pixel lights up on the screen. In high res the display screen is divided into 280 units across and 160 down.

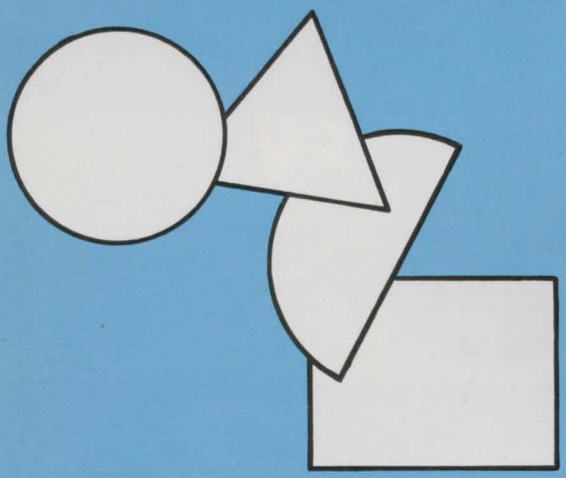
You can use the "HPLOT" command to draw a line made up of many pixels. Type this:

```
30 HPLOT 0,0 TO 279,159
```

Run the program now. We have now drawn a line of pixels from the top left corner of the screen to the bottom right corner.

There are many things you can do in the different graphics modes that are available on the computer. The game "Appleflight," developed in this book, is done in high resolution graphics; but rather than explain all the features of this mode now, let's get on with the game and learn the details as we go along.

SHAPE TABLES

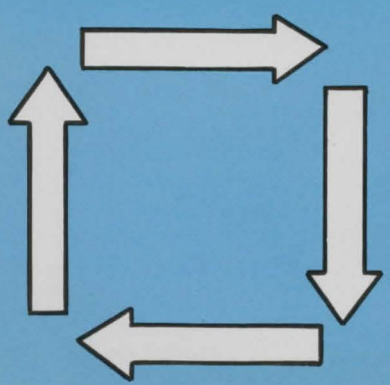


The Apple allows you to draw and store many of the different kinds of shapes you may use in your games program. The computer simply files them away in a shape bank or "shape table." You must assign each shape a number, then when you want one of your shapes to be drawn any place on the screen, you just tell the computer to put shape number such and such at spot such and such.

Unfortunately, the way to draw or define shapes for use in a table is different from drawing shapes using the HPLOT method that we already know. Using this section of the book will make the creation of a shape table easy. Shape table commands can only be used when the computer is in high resolution graphics mode.

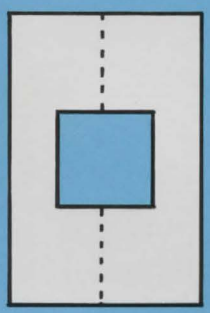
OK, you've drawn a shape on paper and you want to put it into a shape table. Here's what you do: First, break it up into vectors; short lines which point up, down, left or right. We tell the computer these vectors and it follows them like a map. To make a square 2 pixels across and 2 pixels down, you would give the computer these instructions:

plot right,plot right,plot down,plot down,plot left,plot left,plot up,plot up



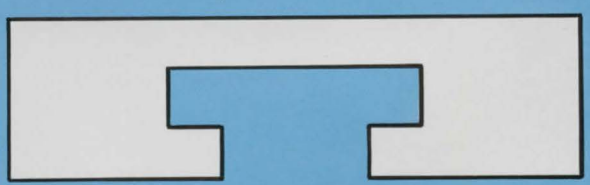
Now suppose you want to have a shape with a hole in it on the screen. You will have to move from one side of the hole to the other to draw both halves of the shape. However, you must tell the computer not to display the connecting lines, or put another way, you tell the computer to draw the connecting lines invisibly. To do this you must tell the computer to MOVE without plotting. These are the type of commands you use when explaining to the computer how to draw your shape.

PLOT UP
DOWN
LEFT
RIGHT



MOVE UP
DOWN
LEFT
RIGHT

See if you can figure out which directions you would have to give to the computer so it can draw this shape:



You can put up to 255 shapes into each shape table. When you want a particular shape, you simply tell the computer the name of the shape table it is in and the number of the shape you want. Now you can have your shape any time you want it and any place you want it with a single command. For example: DRAW 1 AT 100,150 draws the first shape in the table starting at the point which is 100 pixels across and 150 pixels down the screen. If you need to draw one shape many times at different places on the screen, then shape tables are much easier to use than lots of HPLOT commands.

There are many things you can do to shapes, like change their size and color, but we'll look at those things a little later.

The long program on the next few pages is used to create a shape table. You'll have to type in the program very carefully. Then run the program and you will find that it asks you these questions:

1. Is this a new shape table? Y-Yes N-No:
Answer "Y" (plus RETURN) for yes if you are starting a new shape table. If you have already used this program to start a shape table but still have some shapes to add to it, load those shapes in from your diskette by answering "N" (plus RETURN).

2. MOVE:U/D/L/R?E<END>?B<ERASE>?P<PLOT ON/OFF>
Now your drawing screen is in front of you and you can make your shape by typing U for up, D for down, L for left, and R for right. B will erase the shape one dot at a time from the end to the beginning of the shape. P turns the plot off when you want to move without drawing. Striking P again will turn the plot function back on. When your shape is finished, type E to say so and continue. Notice how the program keeps track of the number of vectors you use.

There is one limitation to "move without plotting" commands. If you enter three or more "up" moves with the plot turned off, there is a good chance that the shape will end right there, even if you keep adding vectors. The Apple has this peculiar way of recording vectors, and we can't do anything about it so we just have to avoid using too many "up without plotting" moves.

3. Any more shapes? Y-YES N-NO:
The shape you just finished has been stored. If you answer "Y", you are back at the step 2 and you can continue with another shape. "N" here means you stop here and you go to step 4.

4. Save this table? Y-YES N-NO:
Once you have chosen to stop drawing you must tell the computer you want to save the shapes on a disk. If you type "N" for no to this question, the program will end and all your hard work will be lost. Use "N" if you don't want to keep the shapes you have just drawn. If you type "Y" for yes you will see "ENTER SHAPE TABLE NAME:" "SHUTTLESHAPES" is the name we have chosen for the Appleflight game. The computer will save the shapes onto the disk. BE SURE TO HAVE A DISK WITHOUT A WRITE PROTECT TAB IN THE DISK DRIVE!

This program lets you work on up to 5 shapes (in order 1 through 5) to go into one shape table. You can buy other programs which allow you to work on up to 255 shapes in any order you like. They may also allow you to merge shapes from different shape tables into a new table. But for your first experiments with shapes, use this program to enter the shapes we need for the Appleflight game. The draw commands are listed here.

SHAPE #1
(the shuttle):
LLLULDLULDL
ULDLURLLDLL
LULULULULUL
LL

SHAPE #2
(the shuttle rotated):
LULULULULUL
ULUUUUUUUU
ULULULULL

SHAPE #3
(the base):
URURURURUR
URURRRRRRR
RRURURURUR
URURURRDD
DDDDPRRRR
RRRRPUUUUU
URDRDRDRD
RDRDRDRRR
RRRRRDRDR
DRDRDRDRD
RDR

SHAPE #4
(the explosion):
LULULUURRD
RDRRUUURD
RDRDRRURU
RURDRDDDD
LLLLLLLLLL

SHAPE #5
(the turrets):
RRRRRRRUUU
UULLDDDLLL
UUULLDDDD
D

Save these shapes on your disk with the name SHUTTLESHAPES.

SHAPE TABLE PROGRAM

```
NEW
100 D$ = CHR$ (4)
110 DIM S1%(100),V%(5,100),L%(5)
120 BEG = 24576
130 TEXT : HOME : VTAB 10: PRINT "IS THIS A NEW SHAPE TABLE?"
140 PRINT : PRINT : INPUT "Y-YES N-NO: ";A$
150 IF A$ = "N" THEN GOSUB 5000:SH = SH + 1: GOTO 199
160 IF A$ < > "Y" THEN GOTO 130
170 NS = 5
190 SH = 1
199 IF SH = > 6 THEN PRINT : PRINT : PRINT "TABLE HAS FIVE
    SHAPES": FOR I = 1 TO 750: NEXT : GOTO 130
200  HOME:  VTAB 10:  PRINT "TO START SHAPE# ";  SH;  " PRESS
    RETURN ";
210 GET Q$
220 PRINT : IF ASC (Q$) <  > 13 THEN GOTO 200
230 X = 100:Y = 100
240 HGR : HCOLOR= 3: SCALE= 1: ROT= 0
250 HPLOT X,Y
260 I =0:Q = 0:PI = 4:C = 0
270 IF I < 0 THEN I = 0
280 IF I = 99 THEN S$ = "E": GOTO 300
290 V = I: GOSUB 1000
300 IF S$ = "E" THEN HOME :  TEXT :  VTAB 10:  PRINT  "SAVING
    SHAPE":  GOTO 330
310 IF S$ = "B" THEN GOSUB 3000: GOTO 270
320 S1%(I) = S:I = I + 1: GOTO 270
330 FOR K = 0 TO I
340 IF C = 2 AND S1%(K) > 0 AND S1%(K) < 4 THEN 370
350 IF C < 2 THEN 370
360 C =  0:Q = Q + 1
370 V%(SH,Q) = V%(SH,Q) + S1%(K) * (8 ^ C)
380 C = C + 1
390 NEXT K
400 L%(SH) = Q
410 GOSUB 2000
420 IF SH = NS THEN GOTO 460
430 TEXT :  HOME :  VTAB 10: INPUT "ANY MORE SHAPES? Y-YES N-
    NO: ";A$
440 IF A$ = "Y" THEN SH = SH + 1: GOTO 200
450 IF A$ <  > "N" THEN GOTO 430
460 PRINT : PRINT : INPUT "SAVE THIS TABLE? Y-YES N-NO: ";A$
470 IF A$ = "Y" THEN GOSUB 4000: GOTO 490
480 IF A$ <  > "N" THEN GOTO 460
490 END
1000 VTAB 21: PRINT "VECTOR ";V + 1; ":";
1010 PRINT "MOVE: U/D/L/R?E<END>?B<ERASE>?P<PLOT ON/OFF>: ";
     :GET S$
1020 PRINT
1030 IF S$ = "U" THEN S = 0:Y = Y - 1
1040 IF S$ = "R" THEN S = 1:X = X + 1
1050 IF S$ = "D" THEN S = 2:Y = Y + 1
1060 IF S$ = "L" THEN S = 3:X = X - 1
```

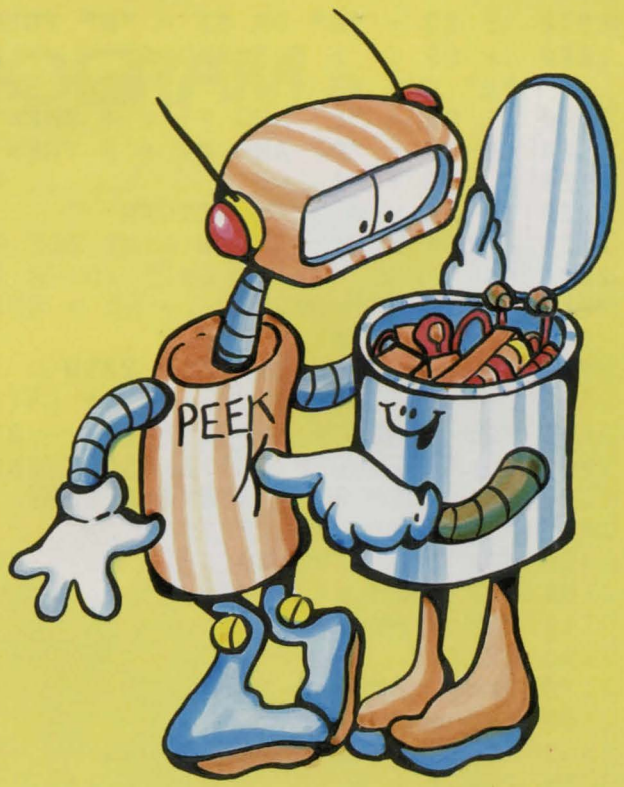
```
1070 IF S$ = "E" OR S$ = "B" THEN RETURN
1080 IF S$ <  > "U" AND S$ <  > "D" AND S$ <  > "L" AND S$ <>
     "R" AND S$ <  > "P" THEN GOTO 1000
1090 IF S$ = "P" AND PI = 4 THEN PI = 0:GOTO 1000
1100 IF S$ = "P" AND PI = 0 THEN PI = 4:GOTO 1000
1110 S = S + PI
1120 IF PI = 0 THEN RETURN
1130 HPLOT X,Y: RETURN
2000 POKE BEG,NS
2010 POKE BEG + 1,SH
2020 OFF(1) = NS * 2 + 2
2030 IF SH = 1 THEN GOTO 2050
2040 OFF(SH) = OFF(SH - 1) + L%(SH - 1) + 2
2050 MEM = BEG + 2 + 2 * (SH - 1)
2060 POKE MEM,OFF(SH) - 256 * INT (OFF(SH) / 256)
2070 POKE MEM + 1, INT (OFF(SH) / 256)
2080 MEM = BEG + OFF(SH)
2090 FOR J = 0 TO L%(SH)
2100 POKE MEM,V%(SH,J)
2110 MEM = MEM + 1
2120 NEXT J
2130 POKE MEM,0
2140 RETURN
3000 IF I < = 0 THEN RETURN
3010 I = I - 1
3020 HCOLOR= 0
3030 HPLOT X,Y
3040 IF S1%(I) = 4 OR S1%(I) = 0 THEN Y = Y + 1
3050 IF S1%(I) = 5 OR S1%(I) = 1 THEN X = X - 1
3060 IF S1%(I) = 6 OR S1%(I) = 2 THEN Y = Y - 1
3070 IF S1%(I) = 7 OR S1%(I) = 3 THEN X = X + 1
3080 HCOLOR= 3
3090 RETURN
4000 TEXT : HOME : VTAB 10: INPUT "GIVE SHAPE TABLE A NAME: "
     ;NAME$
4010 LGTH = OFF(SH) + L%(SH) + 3
4060 PRINT D$"BSAVE";NAME$;",A$6000,L";LGTH
4070 RETURN
5000 HOME : VTAB 10: INPUT "ENTER TABLE NAME: ";NAME$
5010 PRINT D$"BLOAD";NAME$
5020 NS = PEEK (BEG)
5030 SH = PEEK (BEG + 1)
5035 IF SH > = 5 THEN RETURN
5040 L%(SH) = 0
5050 OFF(SH) = PEEK (BEG + 2 + 2 * (SH - 1)) + 256 * PEEK (BEG
     + 2 + 2 * (SH - 1) + 1)
5060 MEM = BEG + OFF(SH)
5070 A = PEEK (MEM): IF A <  > 0 THEN L%(SH) = L%(SH) + 1:MEM=
     MEM + 1: GOTO 5070
5080 IF SH = 1 THEN GOTO 5130
5090 NUM = SH
5100 L%(NUM - 1) = OFF(NUM) - OFF(NUM - 1)
5110 IF NUM - 1 = 1 THEN GOTO 5130
5120 NUM = NUM - 1: GOTO 5100
5130 RETURN
```

Save the shape table program as SHAPE PROGRAM.

PEEK AT A POKE

Look closely at the shape table program you just typed in. Two of the BASIC commands, PEEK and POKE, do more than first meets the eye. Strictly speaking, these commands allow you to look at (PEEK) and change (POKE) the values that are held in your computer's memory. These commands can be used for some fancy programming tricks. For example, in the next few pages you'll see how to put a music routine into memory using POKE commands. Other POKE commands will help us display graphics. The PEEK commands are especially useful during games when, as the programmer, you must find out which key is being pressed or which game paddle is being used.

Here are the PEEKs and the POKEs that will help you to understand the Appleflight game and to write your own games. Don't bother trying to memorize them, just peek at this page when you want to poke some fancy stuff into your program.



POKE −16304,0	CALLS UP GRAPHICS MODE
POKE −16303,0	CALLS UP TEXT MODE
POKE −16302,0	FULL SCREEN GRAPHICS; NO TEXT ON SCREEN
POKE −16301,0	MIXED SCREEN GRAPHICS; GRAPHICS AND TEXT
POKE −16300,0	DISPLAYS GRAPHICS PAGE 1
POKE −16299,0	DISPLAYS GRAPHICS PAGE 2
POKE −16298,0	LOW RESOLUTION GRAPHICS MODE
POKE −16297,0	HIGH RESOLUTION GRAPHICS MODE
PEEK (−16384)	READS THE KEYBOARD TO SEE IF A KEY HAS BEEN PRESSED
POKE −16368,0	GETS THE KEYBOARD READY TO READ IN THE NEXT CHARACTER
PEEK (−16336)	BEEPS THE SPEAKER
POKE 230,32	DRAWS ON GRAPHICS PAGE 1
POKE 230,64	DRAWS ON GRAPHICS PAGE 2

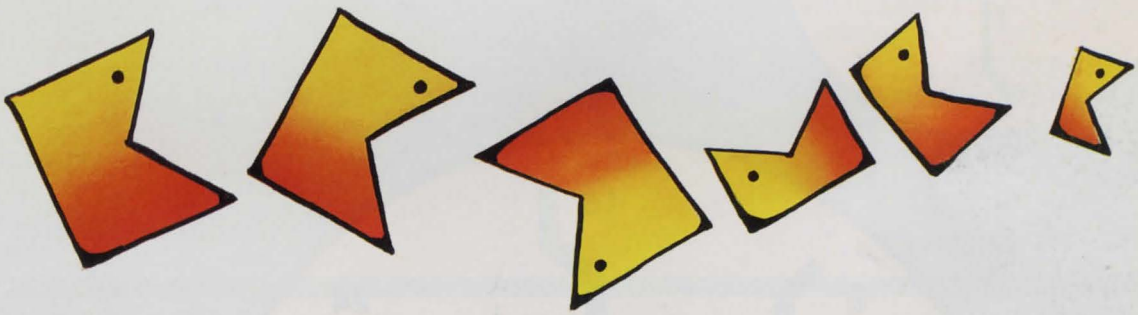
COLOR

Type in this program being sure to copy the line numbers exactly. This is the first part of the Appleflight game and the line numbers become important as we add other parts.

```
NEW
4200 D$=CHR$ (4)                    special control character
4225 PRINT D$"BLOAD SHUTTLESHAPES,   loads in shape table
     A$6000"
4250 POKE 232,0:POKE 233,96         tells the computer where to find shapes in table
4300 HGR                            sets up high resolution graphics mode
4325 POKE -16304,0:POKE -16302,0    full screen /hi res graphics
4350 SCALE=1:ROT=0                  shape size and rotation
4370 HCOLOR=3                       picks a color; white
4375 DRAW 1 AT 100,100              draws the shuttle
4400 DRAW 5 AT 150,185              draws the turret
5100 END
```

When you run the program you should see the turret and the space shuttle on the screen. Return to text mode and save this program by typing:

```
TEXT
SAVE APPLEFLIGHT
```



Color

If your Apple is hooked up to a color display, like a color television, you can draw shapes in different colors. Retype line 4370 of the last program:

```
4370 HCOLOR=6
```

Run the program again!

Now the shuttle and the turret will appear in blue. If you do this on a display which does not have color, you will notice that the picture stays in the same color but appears to have lost some pixels. This is because the single color screen doesn't know how to display the color you've asked for. From now on, if you have a single color display, use HCOLOR=3 to draw a shape and HCOLOR=0 to get the same color as the screen background.

By changing the value of HCOLOR we can change the color of our shape. In fact, we can draw the shapes in any of six different colors.

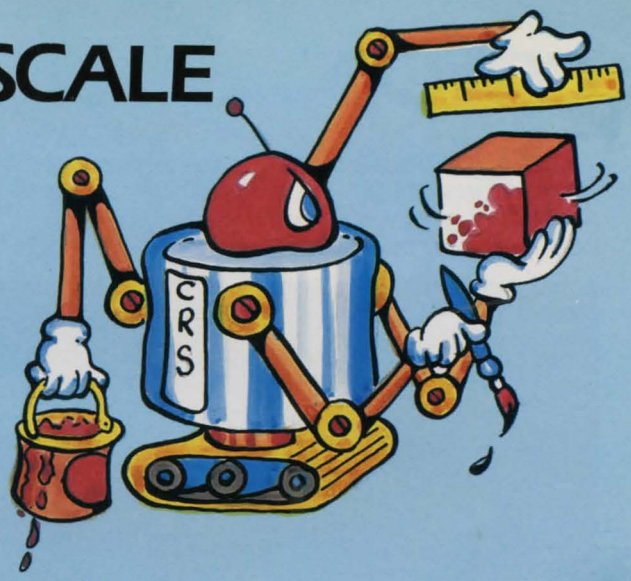
The colors are:

HCOLOR=0	Draws the shape in	black
=1		green
=2		violet
=3		white
=4		black
=5		orange
=6		blue
=7		white

Notice that either 0 or 4 will produce a black shape and that 3 and 7 both draw in white.

Try running the program using all 8 color values. Notice that the shape disappears totally when you use 4 or 0. Do you know why? If you have trouble thinking of a good reason, imagine drawing with a black pen on a piece of black paper.

ROTATION AND SCALE

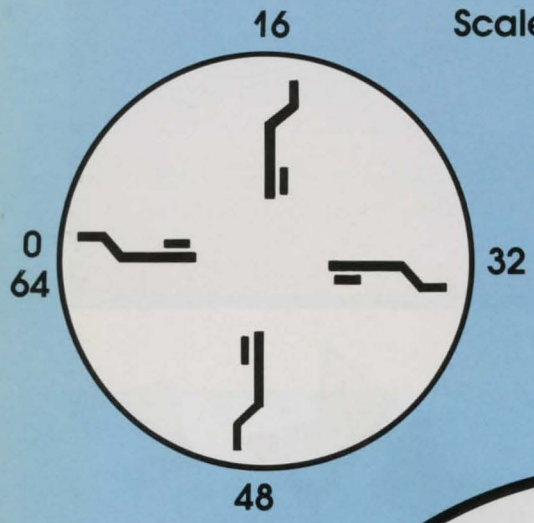


Scale

You can increase the size of the shuttle on the screen by changing the variable SCALE. Try changing line 4350 to read:

4350 SCALE=2:ROT=0

Notice how the same shape appears to wrap around the screen and come out the top. You can try different values of scale from 0 to 255.

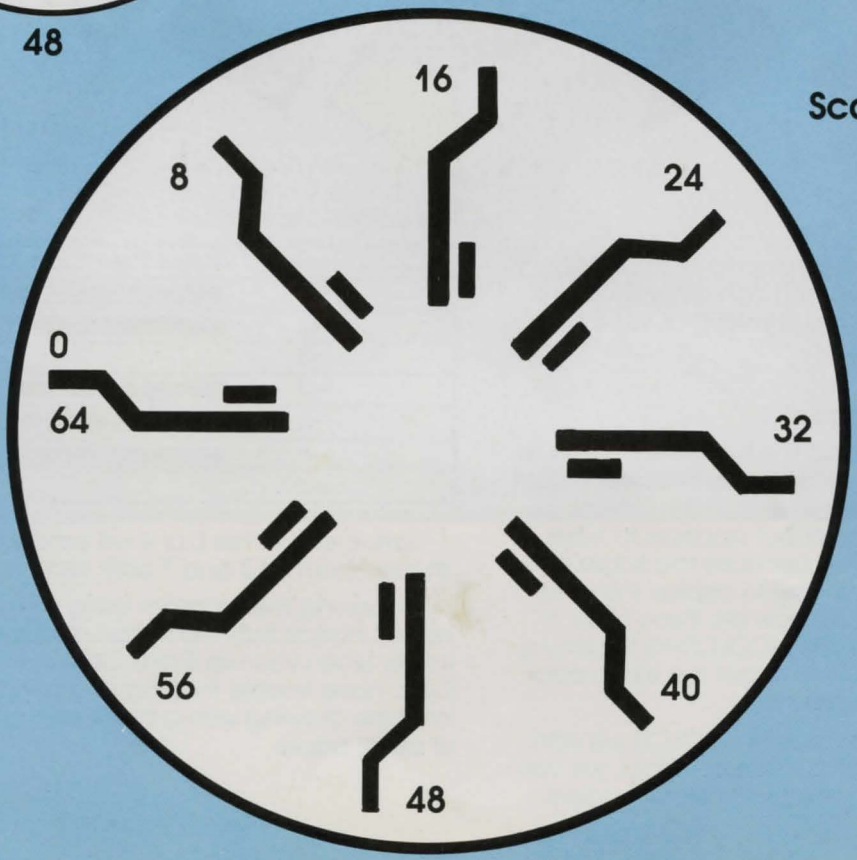


Rotation

Try retyping line 4350 like this:

4350 SCALE=2:ROT=16

Run the program now. The shuttle turns a quarter turn. This is called ROTation. SCALE and ROTation are related; when SCALE=1 you can only rotate using the values 0, 16, 32 or 64 to complete the full circle. When SCALE=2 you can rotate 0, 8, 16, 24, 32, 40, 48, 56, 64 and so on. Try typing in different values of SCALE and ROTation and see what happens.



MOVING RIGHT ALONG

We now know to draw shapes on the screen using different colors, rotations and scales; but unless we learn how to make the shapes move, our games won't be very exciting.

Load in the program from the previous page by typing:
LOAD APPLEFLIGHT

Now type this:
DEL 4370,4400

Next, if you want the shuttle to move, add in these lines:

```
100  GOTO 3950                          jumps to main part of program
3250 IDX=0                              looks at row 0 in table
3375 HCOLOR=0: DRAW 1 AT LAST%          erases at last position.
     (IDX,0),LAST%(IDX,1)
3600 HCOLOR=3:DRAW CURSHAP AT X,Y       draws current shape on screen
3650 LAST%(IDX,0)=X:LAST%(IDX,1)=Y      saves last ship location
3925 RETURN                             returns from subroutine
3950 DIM LAST%(2,8)                     space for keeping old positions
3975 INC=1                              ship speed set to one
4600 CURSHAP=1                          current shape #1 the ship
4675 Y=100:LAST%(0,1)=Y                 initial Y position *
4700 X=20:LAST%(0,0)=X                  initial X position *
4710 FOR I=X TO 250 STEP INC            sets up loop
4715 X=I                                assigns X value to draw
4720 GOSUB 3250                         executes routine to draw shape
4735 NEXT I                             gets a new position to draw
4975 TEXT
```

* These are the values for the horizontal position (X) and the vertical position (Y) on the screen and are called "co-ordinates."

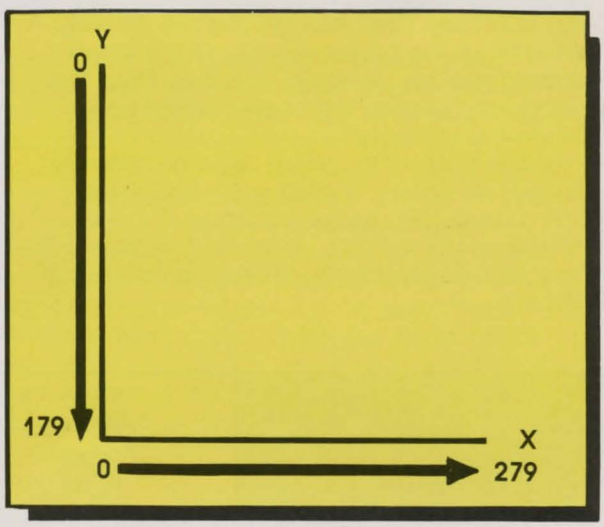
Don't forget to save the program again by typing:
SAVE APPLEFLIGHT

When you run this program the space shuttle moves across the screen. Changing the value of INC (line 3975) to 2 or 4 will make the ship appear to go faster. Try it! (Simply re-type the line – with the line number! – save Appleflight and run it again.)

The first step in making the shape move is to draw it in one place on the screen. Then you erase it and redraw the same shape in a slightly different position. Because the shape is being erased and redrawn so very fast, it appears to be moving. This is called "animation" and is the technique artists use to make cartoons. Motion pictures are, after all, a series of individual pictures or frames.

The variable LAST% is called an array or table. In this game, we use the array to store old variables. For example, in order to erase the ship, we draw over the LAST% position using the background color. Therefore, we save the old X in LAST% (IDX,0) and the old Y in LAST% (IDX,1).

X and Y positions give you co-ordinates.



SMOOTHER SAILING

When you have to draw a lot of shapes it will take the Apple a longer time to draw all the necessary dots. The result is a slight time lag between the frames, giving the motion a jerky appearance. Load in the game so far and make these changes to it:

```
3250 IDX=WPAGE-1                    gets page flipping index
3275 POKE -16299-IDX,0              displays a page
3300 POKE 230,32*WPAGE              draws on a page
3325 WPAGE=WPAGE+1:IF WPAGE=3       get next page
       THEN WPAGE=1
4300 HGR2:HGR
4375 WPAGE=1
4675 Y=20:LAST%(0,1)=Y:LAST%(1,1)=Y
4700 X=100:LAST%(0,0)=X:LAST%(1,0)=X
```

Save the new program as "APPLEFLIGHT" and then run it. The motion of the ship is smoother. At this point the change may not seem too dramatic; however, these lines do help to make the ship move smoothly, especially when the game gets complicated.

Page (Wpage)

This program uses two areas of memory called "pages" to store the display screen. When you do "page flipping," you display one page on the screen while you draw on the one you can't see. Even though you are drawing something, the person looking at the screen doesn't see any changes. You then "flip" the pages and display the one on which you made the changes.

Notice that although we draw on different pages we haven't added any new DRAW commands. We use the variables IDX and WPAGE and the POKE commands in line 3275 and 3300 to keep track of and control all the flipping.

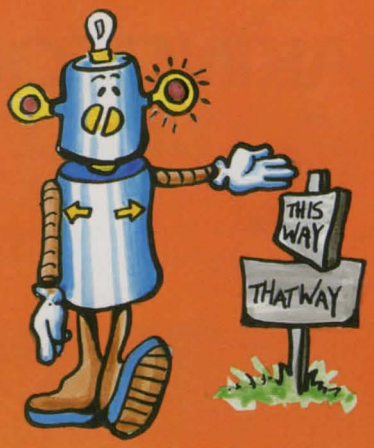
The WPAGE variable keeps track of the page that we are working on (WorkingPAGE).
WPAGE = 1 means that we are working on page 1
WPAGE = 2 means that we are working on page 2

Line 3325 flips WPAGE between these two values when we flip the pages. If WPAGE is 1 then this line makes WPAGE 2 and if WPAGE is already 2 then this line makes WPAGE 1. Can you see why? It first adds one to get WPAGE=3, then since WPAGE=3 it make it 1.

IDX is always set to 1 less than WPAGE. So if WPAGE is 1 then IDX is 0, and if WPAGE is 2 then IDX is 1. Now we know enough to understand how the pages get flipped. The chart shows you what those POKEs do.

IF WPAGE=1 IDX=0	POKE -16299,0 POKE 230,32	Displays page 2 Draws on page 1
If WPAGE=2 IDX=1	POKE -16300,0 POKE 230,64	Displays page 1 Draws on page 2

TURNING CORNERS



Right now the Appleflight shuttle goes in only one direction. Let's take the first step in making it fly around the screen. Add the following lines to the program (if you just turned on the machine LOAD APPLEFLIGHT) and run it.

```
3350 ROT=LAST%(IDX,3)                    gets old rotation value
3375 HCOLOR=0:DRAW LAST%(IDX,2)          changes color to black
     AT LAST%(IDX,0),LAST%(IDX,1)        erases shape at old x,y position
3550 ROT=R1                              new rotation value
3575 LAST%(IDX,3)=R1                     saves rotation value
3675 LAST%(IDX,2)=CURSHAP                saves shape number
3975 INC=4
4425 LAST%(0,2)=CURSHAP:                 initial shape numbers
     LAST%(1,2)=CURSHAP
4703 XQUIT = 250                         end of screen/turn around
4704 K = INC                             speed and direction across screen
4705 FOR J = Y TO 190 STEP INC           goes down the whole screen
4706 Y = J                               current y value
4710 FOR I=X TO XQUIT STEP K
4741 XQUIT=XQUIT+230:IF XQUIT            gets final x depending on direction
     >250 THEN XQUIT=20
4745 R1=R1+32: IF R1=64 THEN R1          adjusts rotation
     =0
4747 K = -K                              reverses increment depending on direction
4748 NEXT J                              gets next y value
```

(Don't forget to save these changes!)

These changes allow the shuttle to fly back and forth across the screen. Notice that line 4745 rotates the ship so it is facing the other way. If the ship was facing right (R1=0) then it will turn to face left (R1=R1+32=0+32=32) and if it was facing left (R1=32) then it will turn to face right (R1=R1+32=32+32=64:when R1=64 the IF statement changes it to 0).

Since R1 can change we have to remember the rotation of any ship we want to erase. We also added code (program lines) to remember the number of the shape. The shape number and rotation of the ship last drawn on page 1 are held in LAST%(0,2) and LAST%(0,3). The shape number and rotation of this ship last drawn on page 2 are held in LAST%(1,2) and LAST%(1,3).

The rest of the code determines at what position the shuttle should be drawn. We use the variable XQUIT to tell us at what X position (horizontal screen position) to stop. The end of the line is position 250 if you are heading right and position 20 if you are heading left. When the ship moves across the screen it's really moving from position 20 to position 250. To do this, we use a program loop (FOR I =X (20) TO XQUIT (250) STEP INC (4)) which draws the shuttle at every position between 20 and 250 using steps of 4 (20, 24, 28, ...). Next, the shuttle moves down a line and goes in the other direction counting backwards (FOR I = X (250) TO XQUIT (20) STEP INC (−4)).

MAKING SOUNDS

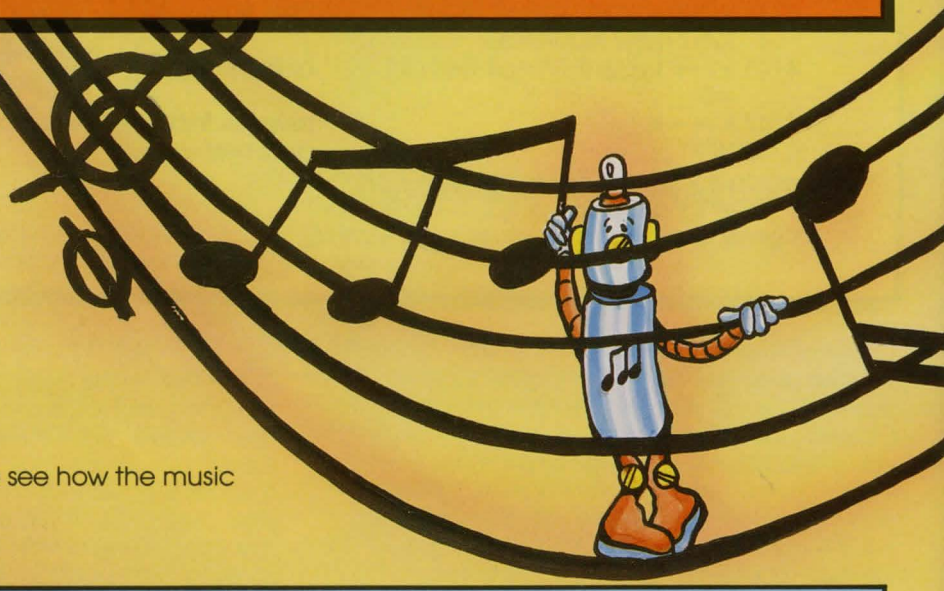
The Apple has the ability to make a whole variety of sounds. The simplest is its characteristic beep, which you can produce by holding down the CTRL key and the letter G at the same time. To see how this same beep can be produced from within a program, type in this:

```
NEW
1 FOR I=1 TO 50          do it fifty times
5 A=PEEK (-16336)        make the beep
10 NEXT I
15 END
RUN
```

When you get bored with this sound and want your games to have more realistic sound effects, you need the machine language program listed here:

```
NEW
LOAD APPLEFLIGHT
5125 ZZ = 300
5150 POKE ZZ,166:POKE ZZ+1,0:POKE ZZ+2,160:POKE ZZ+3,239
5175 POKE ZZ +4,202:POKE ZZ+5,208:POKE ZZ+6,5:POKE ZZ+7,173
5200 POKE ZZ +8,48:POKE ZZ+9,192:POKE ZZ+10,166:POKE ZZ+11,0
5225 POKE ZZ+12,136:POKEZZ+13,208:POKE ZZ+14,245:POKE ZZ+15,198
5250 POKEZZ+16,1:POKE ZZ+17,208:POKE ZZ+18,239:POKE ZZ+19,96
5275 RETURN
```

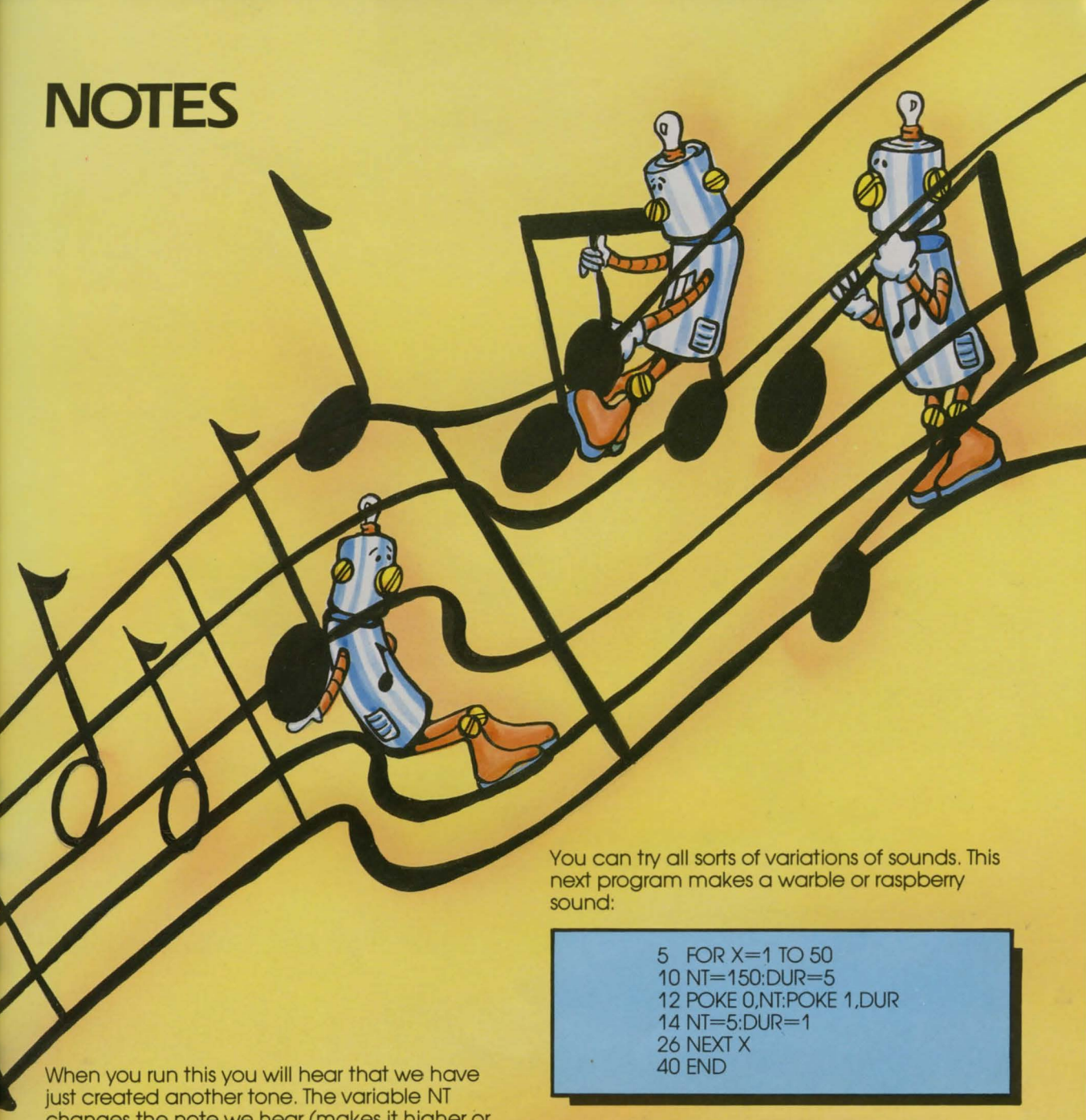
Don't forget to save this program as APPLEFLIGHT.



Now let's take a minute to see how the music program works.

```
Type this:
1  GOSUB 5125          sets up music
5  NT=150              picks a note
10 DUR=50              how long is the note
15 POKE 0,NT
20 POKE 1,DUR
25 CALL R              plays the note
30 END
```

NOTES



When you run this you will hear that we have just created another tone. The variable NT changes the note we hear (makes it higher or lower) and the variable DUR (DURation) tells the computer how long to play it. Either variable can have a value from 1 to 255.

Make these changes to the program and run it:

```
10 DUR=5
14 FOR NT=1 TO 255
26 NEXT NT
```

You can try all sorts of variations of sounds. This next program makes a warble or raspberry sound:

```
 5  FOR X=1 TO 50
10 NT=150:DUR=5
12 POKE 0,NT:POKE 1,DUR
14 NT=5:DUR=1
26 NEXT X
40 END
```

Try some variations yourself to see if you can come up with a good sound for the space shuttle and a neat explosion. There will be suggestions for these later in the book.

To clean up the experiment part of this page, type:

```
DEL 1,40
Save Appleflight.
```

GETTING CONTROL

We've seen that we can make the shuttle go in different directions; the next big step allows you to take control of the shuttle and fly it yourself. Type in the following changes to the Appleflight program. When you run the program the shuttle will fly in the direction it is pointed, just like before, but see what happens when you press the key marked G.

```
DEL 4703,4748
125  REM MAINLINE LOOP
150  GOSUB 1850                    gets keyboard input
200  GOSUB 3250                    draws the shuttle
275  GOTO 150
1850 REM KEYBOARD INPUT
2075 A=PEEK (-16384)               peeks the keyboard
2100 IF A<127 THEN GOTO 2750       no key pressed
2125 POKE -16368,0                 clears keyboard scan
2150 A$=CHR$ (A-128)               figures out which key
2175 IF A$="E" THEN GOTO 4975      ends the game
2200 IF A$="G" THEN R1=R1+16:      rotates one step
     GOTO 2550
2550 REM
2575 IF R1=64 THEN R1=0            full circle..R1 returns to 0
2650 IXD%=R1/16                    index to get x, y, shape number, etc.
2675 XINC=RM%(IXD%,0):YINC=RM%     gets x increment, y inc.
     (IXD%,1)
2700 CURSHAP=RM% (IXD%,2)          gets current shape number
2725 R1=RM%(IXD%,3)                gets new R1
```

ROTATION

```
2750 REM COME HERE IF NO CHANGE
3100 X=X+XINC:Y=Y+YINC          gets new x, y
3125 IF X>279 THEN X=0          checks for screen edges
3150 IF Y>190 THEN Y=0
3175 IF X<0 THEN X=279
3200 IF Y<0 THEN Y=190
3225 RETURN
3950 DIM RM%(8,4),LAST%(2,8)
4000 FOR I=0 TO 3               loads in rotation guide table
4025 FOR J=0 TO 3
4050 READ A%
4075 RM%(I,J)=A%
4100 IF J=0 OR J=1
       THEN RM%(I,J)=A%*INC
4125 NEXT:NEXT
4150 DATA 1,0,1,0,0,1,1,16,
       -1,0,1,32,0,-1,1,48
4575 XINC=RM%(0,0):YINC=RM%(0,1)   initial xinc, yinc
4950 GOTO 125                   go to main program
(Save first, then run.)
```

The shuttle turns 90 degrees and flies in a different direction! If you have an Apple IIe, hold the G key down and watch the shuttle keep turning. The same thing can be done on the Apple II or Apple II+ by holding the G key and pressing the key marked REPT (REPeaT). Try playing with this until you have mastered the controls of the shuttle. When you want to move on, you can end the flight by pressing the key marked E. Notice that now the shuttle flies as long as you want it to and ends when you say so.

HOW WE GOT CONTROL

We added quite a few lines here so let's take a moment to explain, in English, what they all do. First off we've added a part right at the start called "MAINLINE LOOP." A mainline loop is the central controlling part of a program which calls on the other parts of the program to do their stuff.

Line 150 says: Go and find out if a key has been pressed and then figure out where to draw the ship.

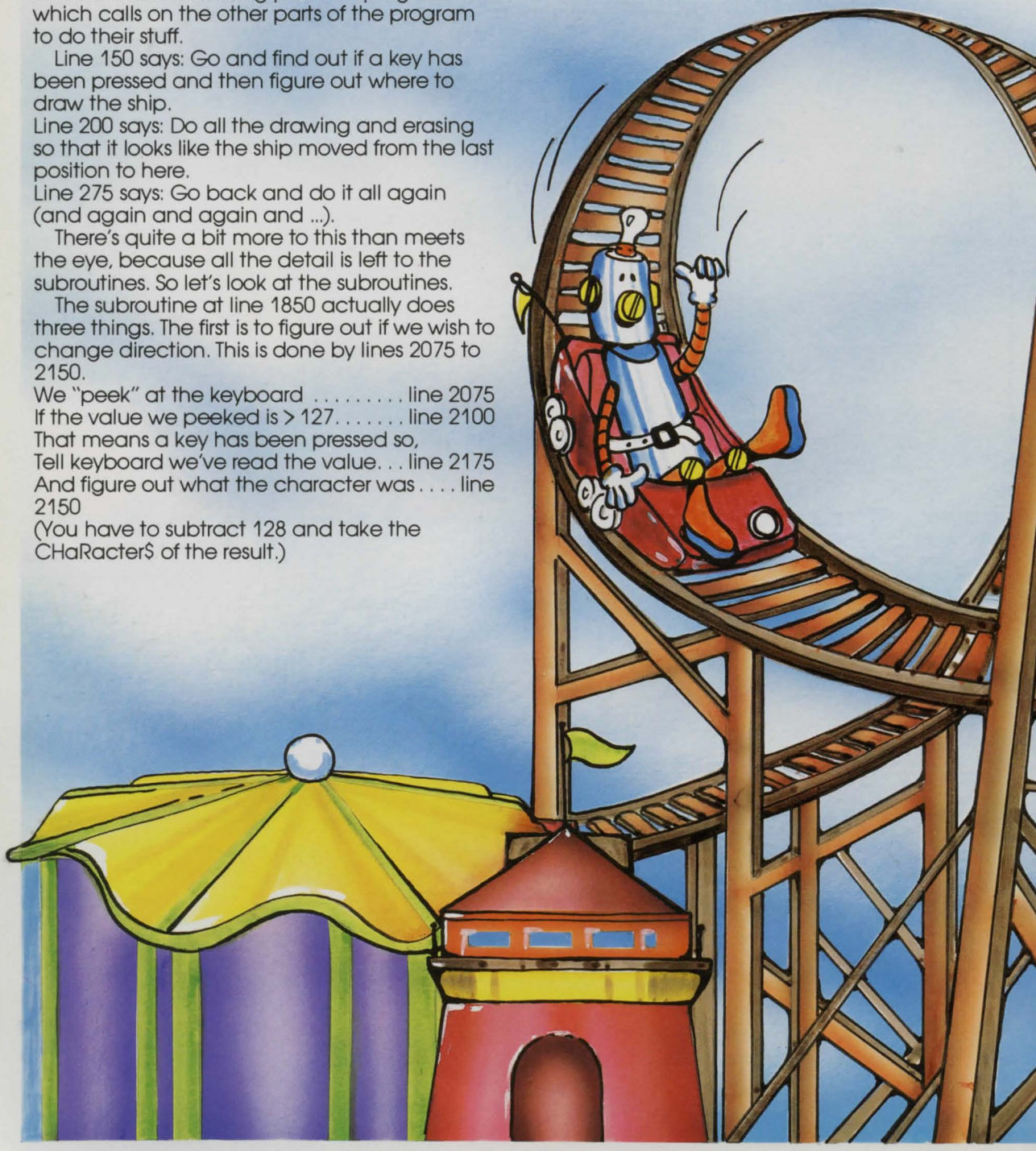
Line 200 says: Do all the drawing and erasing so that it looks like the ship moved from the last position to here.

Line 275 says: Go back and do it all again (and again and again and ...).

There's quite a bit more to this than meets the eye, because all the detail is left to the subroutines. So let's look at the subroutines.

The subroutine at line 1850 actually does three things. The first is to figure out if we wish to change direction. This is done by lines 2075 to 2150.

We "peek" at the keyboard . . . . . line 2075
If the value we peeked is > 127. . . . . line 2100
That means a key has been pressed so,
Tell keyboard we've read the value. . . line 2175
And figure out what the character was . . . . line 2150
(You have to subtract 128 and take the CHaRacter$ of the result.)



The next thing the routine does is to change direction depending on what key was pressed. If the key pressed was E, then the game ends, but if it was G then we add 16 to the rotation to make the shape turn 90 degrees the next time we draw it.

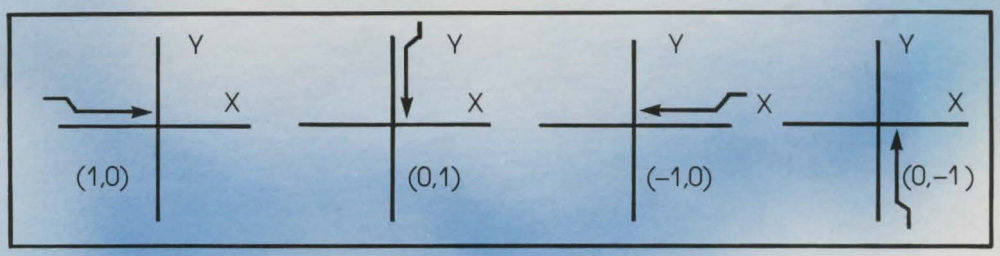
Line 2575 knows that a rotation of 64 is the same as a rotation of 0, and since the program is only concerned with rotations less than 64, the rotation is set back to 0.

Lines 2650 to 3200 determine the where, how and what of drawing the ship. At the start of the game we stored all the information we needed to do this in a table called RM%. Each row or line in this table tells the tale for one particular rotation. There are 4 possible rotations in the table; because the computer recognizes values from 1-64, each line of the table represents 64/4=16 rotations. Line 2650 determines which of the 4 lines in the table to look at by dividing the R1 by 16.

Here is how the table was constructed: RM%

	XINC	YINC	CURSHAP	ROTATION
0	1	0	1	0
1	0	1	1	16
2	−1	0	1	32
3	0	−1	1	48

The values of 0 - 3 in the first column are the result of R1/16.



Now it's easy to modify the program to allow the shuttle to rotate in the opposite direction. Just add the lines below save and run the program. Can you guess what key makes the shuttle turn this way?

```
2275 IF A$="H" THEN R1=R1−16:     turns the other way
     GOTO 2550
2625 IF R1=−16 THEN R1=48
```

The next step in getting the shuttle to move realistically is to allow the controls to be more sensitive. We'll change the program so that each hit of the key results in only a 45 degree rotation rather than the 90 degree rotation that the game presently has. Add the following changes to your program and run it. Play with this and see if there is a problem.


```
2200 IF A$="G" THEN R1=R1+8:GOTO
     2550
2275 IF A$="H" THEN  R1=R1-8:GOTO
     2550
2625 IF R1=-8 THEN R1=56
2650 IXD%=R1/8
4000 FOR I=0 TO 7
4150 DATA 1,0,1,0,1,1,1,8,0,1,1,16,-1,1,1,
     24,-1,0,1,32,-1,-1,1,40,0,-1,1,48,
     1,-1,1,56
```

The problem with SCALE=1 is that the only allowable rotations are multiples of 90 degrees (90, 180, 270 and 0). When we specified ROT=8 (or 24, 40 or 56) we said we wanted a 45 degree rotation (or multiple of 45 which is not a multiple of 90). Unfortunately, the Apple can't draw a shape at these rotations unless SCALE =2, so it does the only thing it can to carry out your instruction and draws your shape with SCALE=2.

To get around the problem for the time being, let's increase the SCALE to 2 by changing the following line and running the program.

```
4350 SCALE =2:ROT=0
```

Now the shuttle stays the same size. Unfortunately, it's twice the size it used to be. This means the advantage of high res's being able to draw very detailed pictures is lost as we will always have to display them at 2 or maybe 3 or 4 times the size they were drawn. To solve this problem, we'll use a trick. Instead of drawing shape 1 (the ship's side view), when we want to rotate 45 degrees, we'll use shape 2, which is just shape 1 rotated 45 degrees (the ship drawn at another angle).

It sounds like the hardest part of this change will be keeping track of which shape is the current shape (CURrentSHAPe) and what shape to draw and erase on each of the graphics pages. But, remember we have the table LAST% that saves the number of the last shape drawn on each page, so this remembering which shape part is a snap. Add the code to display shape 2 if the angle is one of the 45 degree angles that the Apple has trouble with. Don't blink now because you might miss it.

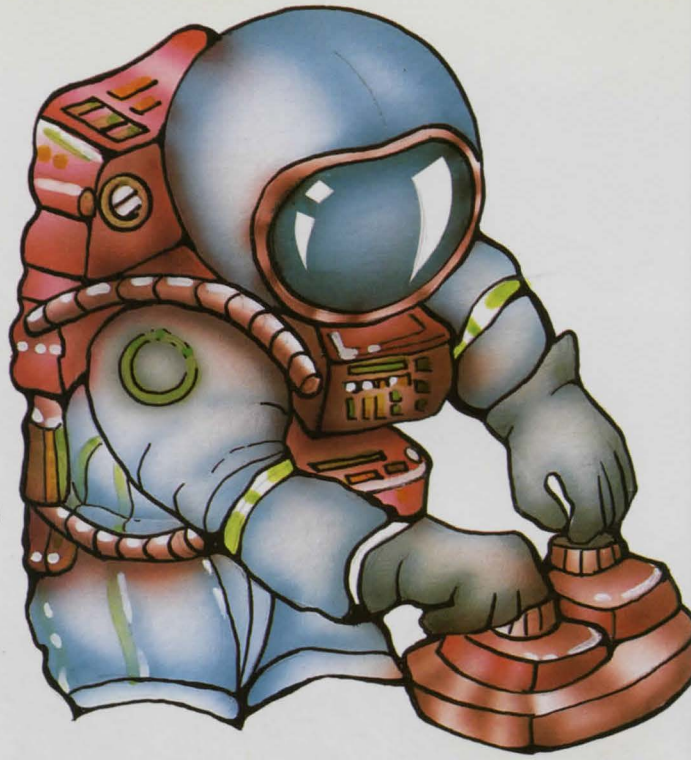
```
4350 SCALE = 1:ROT = 0
2550 IF CURSHAP=2 THEN R1=R1+8
2600 IF R1=72 THEN R1=8
4150 DATA 1,0,1,0,1,1,2,0,0,1,1,16,-1,1,2,
     16,-1,0,1,32,-1,-1,2,32,0,-1,1,48,
     1,-1,2,48
```

When you run this now, the shuttle flies at the 45 degree angle at regular size all the way.

So we can use a number of views of the same shape to get around the limitations imposed by the SCALE and ROTate commands. If you had the patience, you could use the same logic to create several more views of the shuttle to allow for finer and finer control, but remember the more lines stored in the RM% array the more memory your program uses up, leaving less available for your program code and other data values. Also, make sure that the type of game you are building needs multiple views before making all kinds of shapes at all kinds of angles. A game like Pac Man doesn't need those multiple views because the man only travels up, down, right or left; he doesn't move on the diagonal.

Another thing to notice is that we still use the ROT command, and we use it with shapes. This means that shape #2 is the one we use to rotate 135, 225 and 315 degrees, so we don't have to redraw versions of the shuttle for each of these rotations.

GAME PADDLES

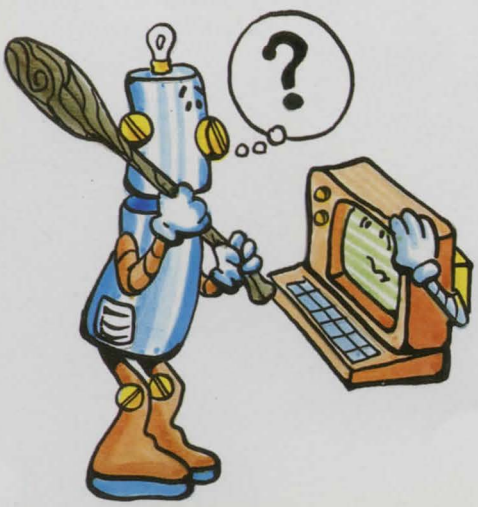


Now that you have mastered controlling the shuttle from the keyboard, let's try game paddle control. (If you don't have a set of game paddles, you will always be able to play Appleflight from the keyboard.)

If you are not sure of how to hook the paddles up, it would be a good idea to ask someone who does know or to follow the manual. On most Apples, connecting the game paddles requires that you lift the lid of the computer and plug the paddles into the socket inside. Remember, SAVE your programs BEFORE turning the computer OFF and ALWAYS turn the computer OFF to plug in the game paddles.

Type the following lines of code to give your game the added feature of using game paddles. If you don't have game paddles, you won't be able to select the games paddles option, but type in this bit of code anyway.

```
1850 IF C$="K" THEN GOTO 2075          selects keyboard
1875 IF C$<>"P" THEN GOTO 2075         if not paddle defaults keyboard
1900 A=PEEK(-16384)                    sees if key has been pressed
1925 IF A>127 THEN GOTO 2750           keyboard overrides paddles
2000 B=PDL(0)                          reads the game paddle
2025 R1=B/32*8                         calculates ship rotation
2050 GOTO 2650
4275 INPUT "CONTROL? P-PADDLES K-KEYBOARD: ";C$
```



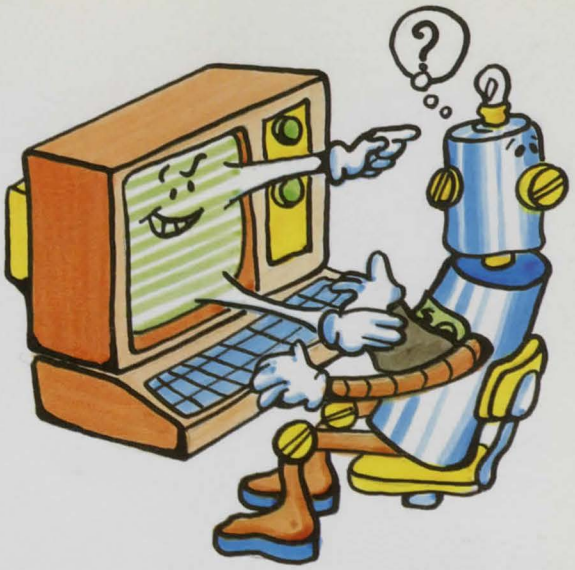
Save this program again as Appleflight and try running it. If you don't have game paddles, run the program to make sure you haven't made any typing errors. If you do have game paddles connected to the computer, then use the one marked "O" to control the rotation of the ship.

Some joysticks will also work with this program. But because the ship rotates in only two directions, only two of the joystick's four positions are used in this game. There are no changes to the code to have your joystick work in this way.

LOOKING BACK

Let's review all that we have done so far. We started off by simply drawing a point on the screen. Then step by step we expanded the program to draw a shape on the screen, our space shuttle. After making the shuttle move, we added control from the keyboard and game paddles. We also learned about making sounds and shape tables. How did we get through all that in so few pages? The secret is that we broke the problem into many small steps. Each step was easy and didn't require too many lines of code. Computer people call this "structured programming."

As you read the rest of the book, keep in mind that all games can be broken down into small, easy-to-manage pieces, making game writing much easier.



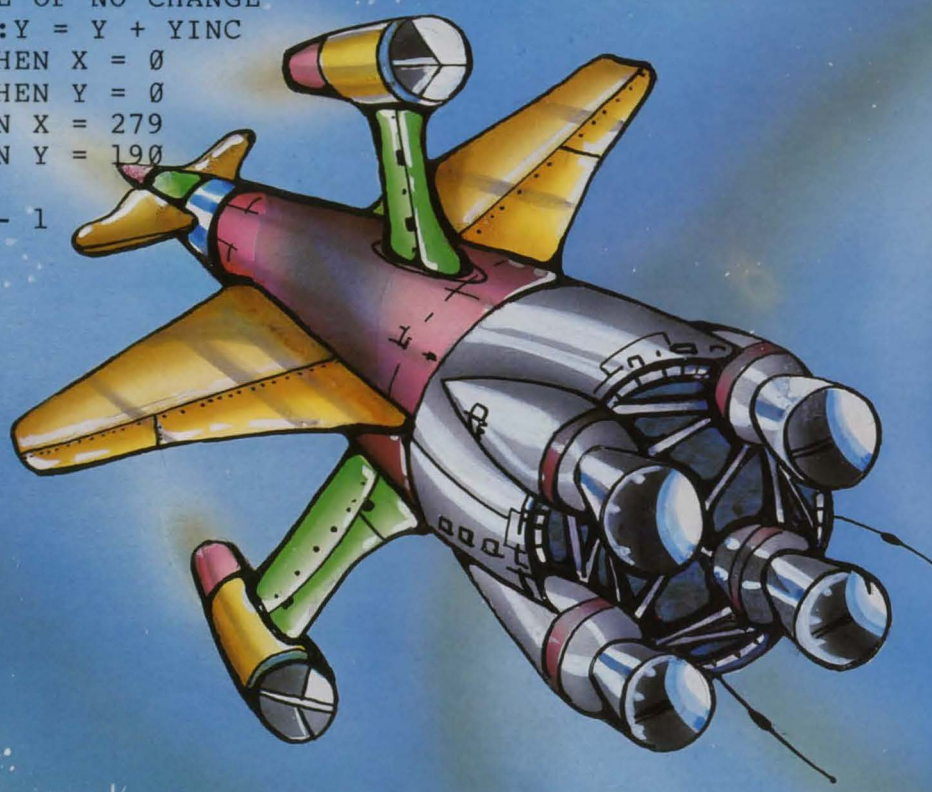
Another good programming habit is to write out what you want to do in English before you write in BASIC. This English program is called "pseudocode," pronounced "sue dough code."

Here's an example of pseudocode for a small BASIC program. Cover the pseudocode column with your hand. Can you understand what the BASIC commands mean? Now cover the BASIC column and read the pseudocode. Chances are that you understand what the pseudocode is trying to explain easier than the BASIC. This is because you understand English more easily than BASIC. Before writing a program in BASIC, you should write it in English. As you modify the English version into smaller and smaller steps, the transfer to BASIC commands becomes easier. Unless you can describe what you want a program to do in English, you won't be able to describe or write the program in a computer language like BASIC.

PSEUDOCODE	BASIC
1. EXAMINE KEYBOARD TO SEE IF ANY KEY WAS PRESSED	10 A=PEEK (−16384)
2. IF A KEY WASN'T PRESSED THEN GO BACK TO 1 AND WAIT FOR IT	20 IF A<127 THEN GOTO 10
3. TELL THE KEYBOARD THAT YOU HAVE READ THE SELECTED KEY	30 POKE −16368,0
4. FIGURE OUT WHICH KEY WAS PRESSED	40 A$=CHR$ (A−128)
5. END THE PROGRAM	50 END

THE GAME SO FAR

```
100 GOTO 3950
125 REM MAINLINE LOOP
150 GOSUB 1850
200 GOSUB 3250
275 GOTO 150
1850 IF C$ = "K" THEN GOTO 2075
1875 IF C$ < > "P" THEN GOTO 2075
1900 A = PEEK (-16384)
1925 IF A > 127 THEN GOTO 2750
2000 B = PDL (0)
2025 R1 = B / 32 * 8
2050 GOTO 2650
2075 A = PEEK (-16384)
2100 IF A < 127 THEN GOTO 2750
2125 POKE -16368,0
2150 A$ = CHR$ (A - 128)
2175 IF A$ = "E" THEN GOTO 4975
2200 IF A$ = "G" THEN R1 = R1 + 8: GOTO 2550
2275 IF A$ = "H" THEN R1 = R1 - 8: GOTO 2550
2550 IF CURSHAP = 2 THEN R1 = R1 + 8
2575 IF R1 = 64 THEN R1 = 0
2600 IF R1 = 72 THEN R1 = 8
2625 IF R1 = -8 THEN R1 = 56
2650 IXD% = R1 / 8
2675 XINC = RM%(IXD%,0):YINC = RM%(IXD%,1)
2700 CURSHAP = RM%(IXD%,2)
2725 R1 = RM%(IXD%,3)
2750 REM COME HERE OF NO CHANGE
3100 X = X + XINC:Y = Y + YINC
3125 IF X > 279 THEN X = 0
3150 IF Y > 190 THEN Y = 0
3175 IF X < 0 THEN X = 279
3200 IF Y < 0 THEN Y = 190
3225 RETURN
3250 IDX = WPAGE - 1
```



```
3275 POKE -16299 - IDX,0
3300 POKE 230,32 * WPAGE
3325 WPAGE = WPAGE + 1: IF WPAGE = 3 THEN WPAGE = 1
3350 ROT= LAST%(IDX,3)
3375 HCOLOR=0: DRAW LAST%(IDX,2) AT LAST%(IDX,0),LAST%(IDX,1)
3550 ROT = R1
3575 LAST%(IDX,3) = R1
3600 HCOLOR= 3: DRAW CURSHAP AT X,Y
3650 LAST%(IDX,0) = X:LAST%(IDX,1) = Y
3675 LAST%(IDX,2) = CURSHAP
3925 RETURN
3950 DIM RM%(8,4),LAST%(2,8)
3975 INC = 4
4000 FOR I = 0 TO 7
4025 FOR J = 0 TO 3
4050 READ A%
4075 RM%(I,J) = A%
4100 IF J = 0 OR J = 1 THEN RM%(I,J) = A% * INC
4125 NEXT : NEXT
4150 DATA  1,0,1,0,1,1,2,0,0,1,1,16,-1,1,2,16,-1,0,1,32,-1,
     -1,2,32,0,-1,1,48,1,-1,2,48
4200 D$ = CHR$ (4)
4225 PRINT D$"BLOAD SHUTTLESHAPES,A$6000"
4250 POKE 232,0: POKE 233,96
4275 INPUT "CONTROLS? P-PADDLE K-KEYBOARD: ";C$
4300 HGR2 : HGR
4325 POKE -16304,0: POKE -16302,0
4350 SCALE= 1:ROT= 0
4375 WPAGE = 1
4425 LAST%(0,2) = CURSHAP:LAST%(1,2) = CURSHAP
4575 XINC = RM%(0,0):YINC = RM%(0,1)
4600 CURSHAP = 1
4675 Y = 20:LAST%(0,1) = Y:LAST%(1,1) = Y
4700 X = 100:LAST%(0,0) = X:LAST%(1,0) = X
4950 GOTO 125
4975 TEXT
5100 END
5125 ZZ = 300
5150 POKE ZZ, 166:POKE ZZ+1,0:POKE ZZ+2,160:POKE ZZ +3,239
5175 POKE ZZ+4,202:POKE ZZ+5,208:POKE ZZ+6,5:POKE ZZ+7,173
5200 POKE ZZ+8,48:POKE ZZ+9,192:POKE ZZ+10,166:POKE ZZ+11,0
5225 POKE ZZ+12,136:POKE ZZ+13,208:POKE ZZ+14,245:POKE ZZ+15,
     198
5250 POKE ZZ+16,1:POKE ZZ+17,208:POKE ZZ+18,239:POKE ZZ+19,96
5275 RETURN
```

SHOOT 'EM UP APPLE

The next task is to arm the shuttle. We'll do this by adding a photon blaster to the ship's arsenal. Let's break this into stages and see what's involved.

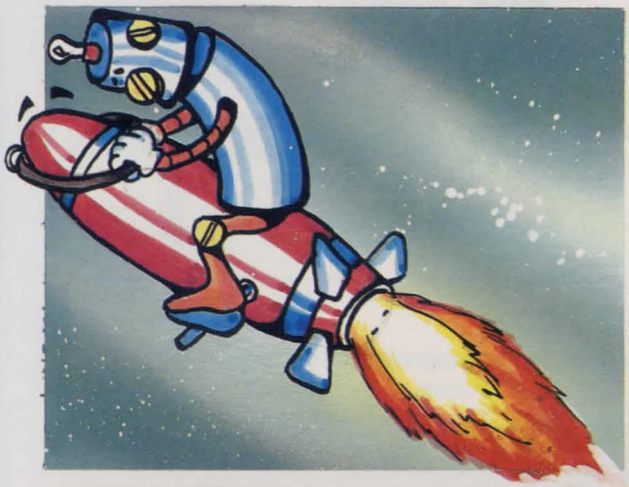
The first stage will be to try to plot the path of the photon when the key is pressed. (When the firing mechanism is activated.) We'll use the "S" key as the "fire" button. Add the following lines to your program so far and run it. What do you notice about how the blasts appear? Be sure to fire and to test turning the ship to make sure everything still works.

```
1950 S=PEEK(-16287)                          checks the paddle shot
1975 IF S>127 THEN GOTO 2375                 shot detected
2350 IF A$<>"S" THEN GOTO 2750               no shot
2375 REM SHOT ROUTINE
2425 S3INC=XINC*3:S4INC=YINC*3               shot x and y increments
2450 S1X=X:S2Y=Y                             shot position
2475 HPLOT S1X,S2Y                           draws the shot
2750 S1X=S1X+S3INC:S2Y=S2Y+S4INC             next shot position
3000 IF S1X>279 OR S1X<0 THEN                edge of the screen reached
       S3INC=0:S4INC=0:S1X=0:
       S2Y=0
3025 IF S2Y>190 OR S2Y<0 THEN
       S3INC=0:S4INC=0:S1X=0:S2Y=0
3625 HPLOT S1X,S2Y                           draws the shot
4725 S1X=0:S2Y=0                             no shot to start with
```

It seems that our shoot routine has a bit of a problem. The shuttle shoots the photon blast but the path of the blast flashes on the display screen. The problem here is that although the path of the blast is being plotted, it is not being erased. So the next step is to erase the previously drawn photon blast. This shouldn't be too difficult since we have already done something very similar with the shuttle itself. Here are the lines to add to the program to fix this problem:

```
2525 LAST%(IDX,4)=S1X:LAST%(IDX,5)=S2Y       saves last shot position
3500 HPLOT LAST%(IDX,4),LAST%(IDX,5)         erases last shot
3900 LAST%(IDX,4)=S1X:LAST%(IDX,5)=S2Y
```

The variable names are getting a little confusing unless you know the reasons for them. The Apple BASIC language only recognizes the first two characters of a variable name.



Type this short program:

```
1 S1X=0
2 S1Y=5
3 PRINT S1X
4 END
```

Run this program and you'll get 5 printed, not the 0. The computer thinks that lines 1 and 2 change the value of the same variable "S1".

(Type:
DELETE 1,4
to make sure that if you added these lines, they don't get mixed up in your game code.)

Because of this problem, variables which refer to shots by the ship begin with an S (S for Shot, clever, eh?) but they are followed by numbers to keep the variables different from each other. Following the number (or second character of the variable) anything that will help us to identify the variable can be added. For example:

S1X	the x position of the shot
S2Y	the y position of the shot
S3INC	how far the shot moves in x direction
S4INC	how far the shot moves in y direction

Maybe now you can see that the variables aren't quite as crazy as they orignally seemed.

Now let's take a closer look at how the shooting routine works.

Line 4725	starts the shot off at 0,0
Line 2350	determines if the S key has been pressed
Line 2375	determines the direction in which to move the shot
Line 2450	says the ship has fired the shot at this point
Line 2475	plots the shot right away
Line 2525	remembers the last position of the shot
Line 2750	determines the next position of the shot
Line 3000, 3025	tells the computer, if the shot has reached the edge of the screen, then don't plot it anymore
Line 3500	erases the last position of the shot
Line 3900	remembers the last position of the shot

We've got one more change to make to perfect the shooting routine. Did you notice what happens when you shoot another shot while the first shot is still on the screen? This is because we planned for one shot at a time when we added the routines to draw and erase the shot. We have to change that program so that the second shot can't go off until the first one has either hit its target or moved off the screen. Add this one line change to fix up the program:

```
2375 IF S1X > 0 OR S2Y > 0 THEN GOTO 2750
```

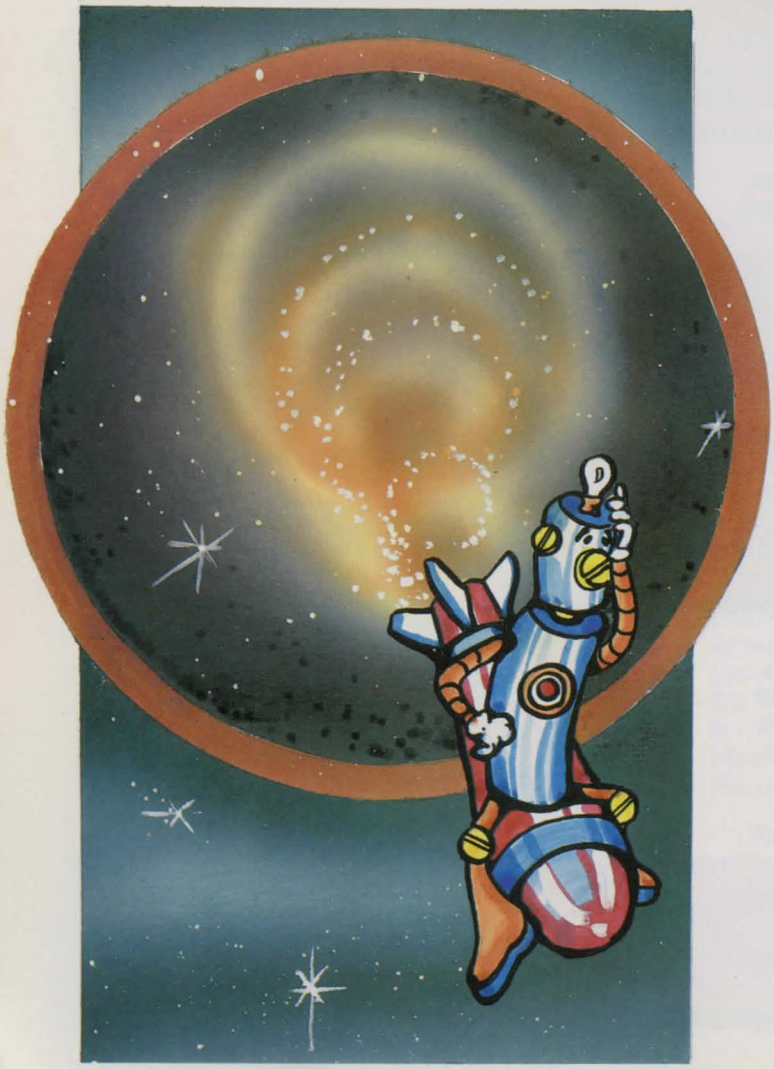
Now when you run the program you get only one shot at a time.

READY AIM

In the next few steps our game will really start to take shape. The following lines, when added to the program, will give the shuttle a target to shoot at. We'll do this in a couple of steps, to show you how to break the task into small sections.

The first step will just draw the target on the screen.

```
4450 L3SHAPE=3:L4SHAPE=4:L5SHAPE=5      sets up names for the shapes
3800 ROT=0:DRAW L3SHAPE AT 150,185      draws the base
```



Run the program and have the shuttle shoot the base. There's a slight problem, isn't there? To the shuttle and its photons, the enemy base isn't even there. They both pass right through!!!

To fix this, the first thing we'll do is to cause the shuttle to crash if it hits the enemy base. This is a pretty complex task; how do you know when the shuttle collides with the enemy base? The easiest way is to use our knowledge of the shapes and their positions on the screen to tell when the shuttle crashes into the enemy base. Once we know how to do this with the shuttle it will be easy to apply the same technique to the photon blasts.

First of all we must find out if the shuttle is close enough for a collision to occur. So, we'll add the following lines. They will determine if the shuttle is being drawn in a place where the enemy base is being displayed. Add these lines and run your program.

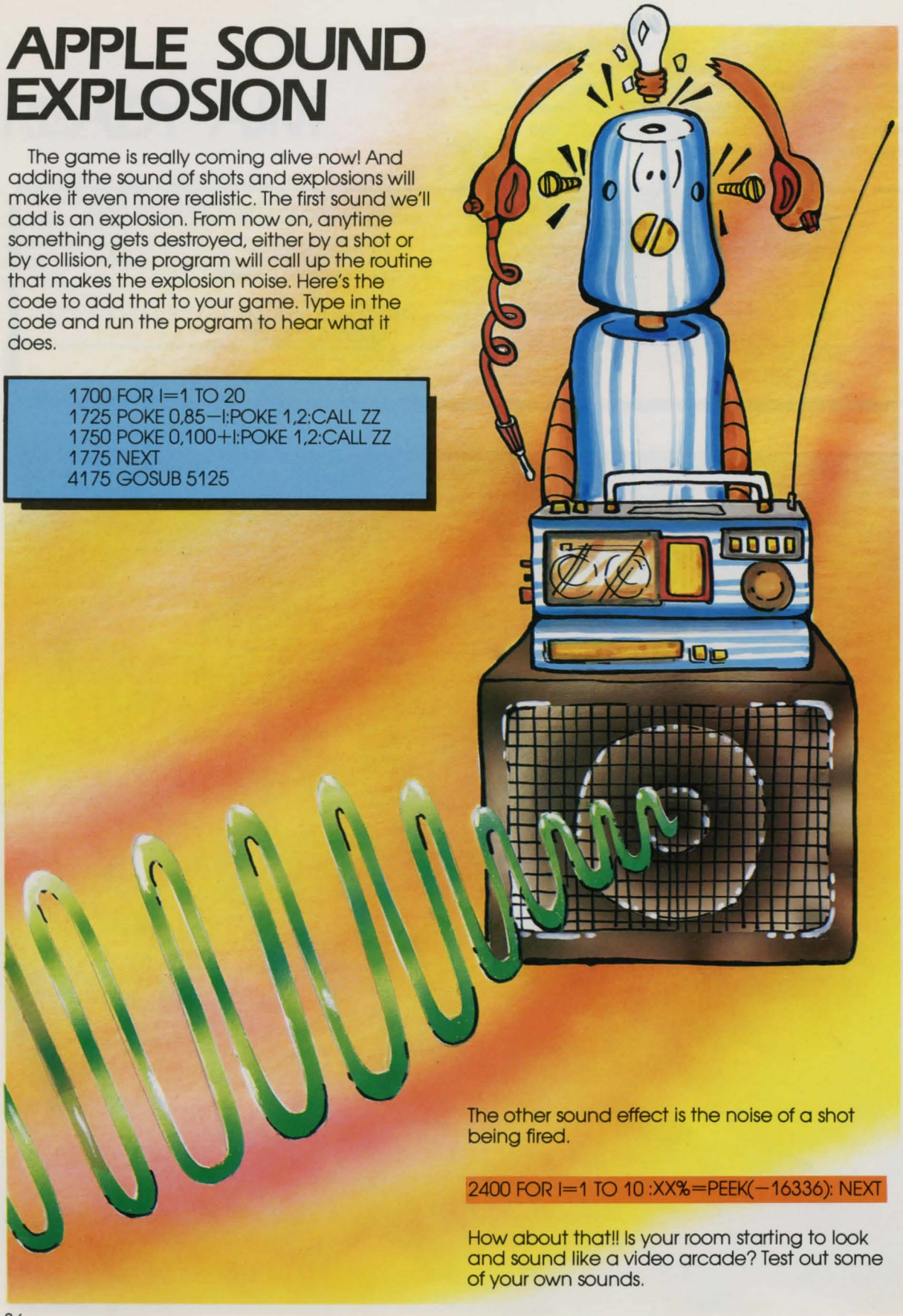
```
225  GOSUB 1525                          the routine to check if ship hit base
1525 REM CHECK FOR COLLISION OF SHIP
1550 IF X<150 OR X>202 THEN RETURN       not close enough
1575 IF Y>177 AND Y<185 THEN GOTO        checks more closely
     1650
1600 IF X<167 OR X>184 THEN RETURN       not close enough
1625 IF Y<166 OR Y>185 THEN RETURN       not close enough
1650 X=100:Y=100                         A CRASH!!!!!
1825 RETURN
```

We estimated the shape of the enemy base to be roughly a square on top of a larger one. This allowed us to determine easily whether or not the shuttle was being drawn inside the enemy base. We could have done this by storing all the points on the enemy base and comparing to see if the shuttle was being drawn over any one. However, this would be a very long program which would have meant that the game would be very slow.

APPLE SOUND EXPLOSION

The game is really coming alive now! And adding the sound of shots and explosions will make it even more realistic. The first sound we'll add is an explosion. From now on, anytime something gets destroyed, either by a shot or by collision, the program will call up the routine that makes the explosion noise. Here's the code to add that to your game. Type in the code and run the program to hear what it does.

```
1700 FOR I=1 TO 20
1725 POKE 0,85-I:POKE 1,2:CALL ZZ
1750 POKE 0,100+I:POKE 1,2:CALL ZZ
1775 NEXT
4175 GOSUB 5125
```



The other sound effect is the noise of a shot being fired.

```
2400 FOR I=1 TO 10 :XX%=PEEK(-16336): NEXT
```

How about that!! Is your room starting to look and sound like a video arcade? Test out some of your own sounds.

TURRETS SIGHTED

Now type in these lines to put the turret (from your shape table) onto the enemy base and into your game.

```
4475 B1X=158:B2Y=177                turret 1 x and y position
4500 B3X=173:B4Y=170                turret 2 x and y position
4525 B5X=188:B6Y=177                turret 3 x and y position
3825 IF BIX > 0 THEN DRAW L5SHAPE   draws the three turrets
       AT B1X,B2Y
3850 IF B3X > 0 THEN DRAW L5SHAPE   at the positions we
       AT B3X,B4Y
3875 IF B5X > 0 THEN DRAW L5SHAPE   just defined
       AT B5X,B6Y
```

Now we have a real target to shoot at; in fact, we have three. Notice that all three are the same shape; we just make the shape appear on the screen in several locations.

The above program lines simply drew the shapes on the screen, they didn't check to see if the shots from the shuttle hit any of them. We'll do that now by checking the first turret.

Add the following lines and run the program. Try shooting at turret 1, the one on the left.

```
175  GOSUB 775                      subroutine to see if shot hit turret
775  IF S1X<158 OR S1X>192          don't check if shot isn't even close
       THEN RETURN
800  IF S2Y > 177 THEN RETURN       shot has gone past turret so don't check
825  IF B1X = 0 THEN GOTO 975       bypass turret if it is already destroyed
850  IF B1X > 165 THEN GOTO 975     shot is past the turret
875  IF S2Y < 173 THEN GOTO 975     shot hasn't reached turret
900  REM A HIT ON TURRET 1          shot has hit turret
925  B1X=0:B2Y=0                    eliminate turret by zeroing position
950  GOTO 1325                      go make explosion noise
975  REM NO HIT ON TURRET 1         the shot missed, better luck next time
1000 RETURN
1325 FOR I = 1 TO 20                make some noise
1350 POKE 0,85-I:POKE 1,2:
       CALL ZZ
1375 POKE 0,100+I:POKE 1,2:
       CALL ZZ
1400 NEXT
1475 S1X=0:S2Y=0:S3INC=0:S4INC=0    the shot is destroyed as well
1500 RETURN
```

Did you notice any problems with this when you ran it? Were you able to destroy the enemy turret? If you had the value of variable INC (line 3975) set at more than 2, you may have had a lot of trouble. Here's why: the shot moves three times further than the ship every time they are drawn (line 2425). If the distance between shot drawings is greater than 7 pixels (if the ship INC is greater than 2) the shot can pass right through the turret.

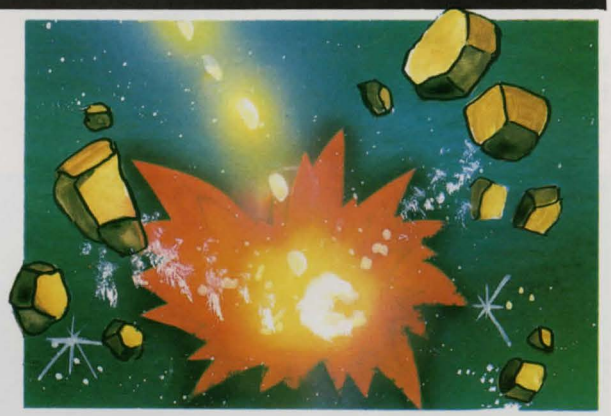
We will overcome this obstacle by using the halfway point between the last shot and the one we just drew, as a checkpoint. For example, if INC is 4, the shot is drawn every 12 pixels and we check every 6 to see whether or not the shot hit the turret. (This method isn't foolproof: if the value of INC is 6, then the shot increment would be 18 pixels and we would check every 9th pixel. Because the turret is only 7 pixels wide, it is possible that we could still miss the turret. To avoid this problem, make sure that the INC is always set at 4 or less.) So, when we're not checking the halfway point an INC of 2 is a problem, but using the halfway point checking method, an INC of up to 4 is OK.

The following changes add the halfway checkpoint to the program. Add them to your program and try to shoot turret 1.

```
775 XCHECK=S1X-S3INC/2:
    YCHECK=S2Y-S4INC/2          finds the halfway point by subtracting
                                half the increment from shot position
780 IF (S1X < 158 AND XCHECK    both must be out of range then we can skip
    <158) OR (S1X>192 AND       check
    XCHECK>192) THEN RETURN
800 IF (S2Y>177 AND YCHECK>177) likewise for the y
    THEN RETURN
850 IF (S1X>157 AND S1X<166)    if shot between right and left sides
    AND (S2Y<178 AND S2Y>172)   and between bottom and top
    THEN B1X=0:B2Y=0:GOTO 1325  then there is a collision
875 IF (XCHECK>157 AND XCHECK   this does the same check for
    <166)                       the
    AND (YCHECK<178 AND YCHECK  halfway point
    >172) THEN B1X=0:B2Y=0:GOTO 1325  halfway point in range, means collision
DEL 925,950                     don't need these lines, delete them
```

Now when we run the program, with an INC of 4, we can destroy the turret and hear the explosion.

We now have a pretty good method of discovering whether or not the shot hit the first turret. We'll add the following lines to check if either of the other two were hit.



```
1000 IF B3X=0 THEN GOTO 1175
1025 IF (S1X>172 AND S1X<181)
     AND (S2Y<171 AND S2Y>164)
     THEN B3X=0:B4Y=0:GOTO 1325
1050 IF (XCHECK>172 AND
     XCHECK<181) AND (YCHECK<171
     AND YCHECK>164) THEN B3X=0:
     B4Y=0:GOTO 1325
1175 REM NO HIT ON TURRET 2
1220 IF B5X=0 THEN RETURN
1225 IF (S1X>187 AND S1X<194)  AND (S2Y<178 AND S2Y>172) THEN B5X=0:B6Y=0:GOTO 1325
1250 IF (XCHECK>187 AND  XCHECK<194)  AND (YCHECK<178 AND  YCHECK>172)
     THEN B5X=0:B6Y=0:GOTO 1325
1275 RETURN
```

The next addition will make the game look a little more realistic. When each turret gets destroyed, it will disappear from the screen, like it's been completely demolished. One method of doing this is to erase all the turrets and then redraw only those that haven't been destroyed. Type this:

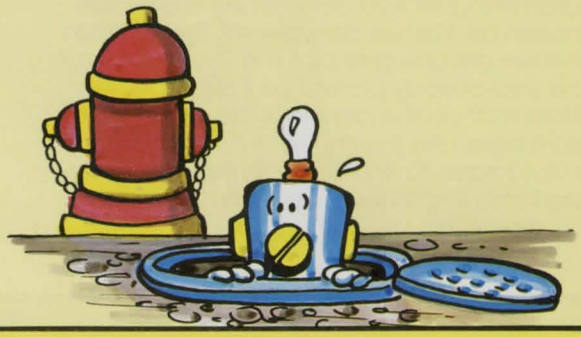
```
3425 ROT=0:DRAW L5SHAPE AT 158,177    erases first turret
3450 DRAW L5SHAPE AT 173,170          erases second turret
3475 DRAW L5SHAPE AT 188,177          erases third turret
```

Now when we run the program we can shoot the turrets which then disappear with the sound of an explosion!

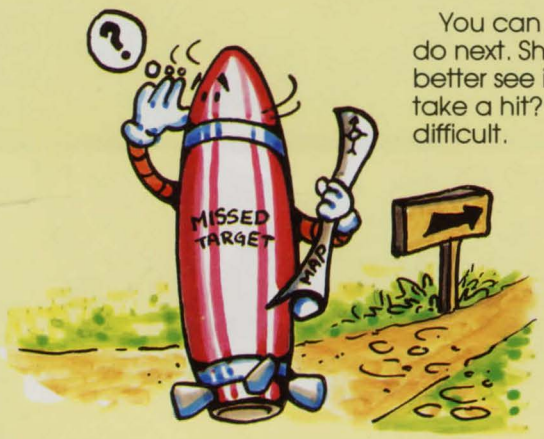
You've probably done pretty good so far, shooting turrets off the base and all, but that's because it's been pretty much of a target practice shoot so far. Wait till you have to dodge enemy fire to shoot at the turrets!!

TAKE COVER

The next step allows the turrets to shoot back at the shuttle. We'll start with the turrets shooting randomly. This means that we will have no way of predicting which of the turrets will shoot or when.

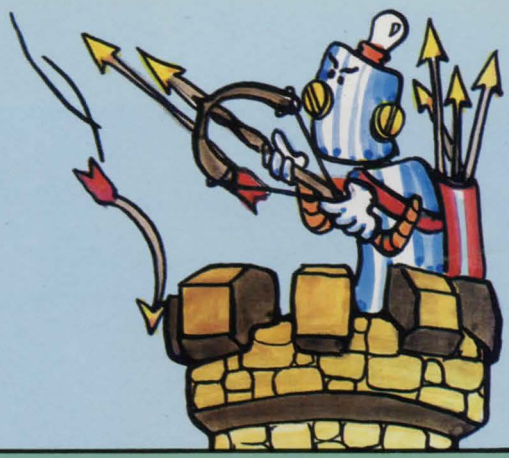


```
4800 SFX=0:SGY=0                          this is the position of the turret's shot
4825 LAST%(0,6)=0:LAST%(0,7)=0            sets up to remember last position
     :LAST%(1,6)=0:LAST%(1,7)=0           of shot on both pages
4850 SDINC=0                              sets initial shot increment to zero
2775 IF SFX > 0 THEN GOTO 2950            don't shoot if shot already fired
2800 LZ = RND (1)                         gets a random number between 0 and 1
                                          and calls it LZ
2825 IF LZ < .5 THEN GOTO 3000            only fires 1/2 the time
2850 IF LZ < .7 AND B1X > 0               if LZ less than .7 and turret 1 exists
     THEN SFX=160:SGY=170:GOTO 2925       fires shot from turret 1
2875 IF LZ <.85 AND B3X > 0               if LZ less than .85 and turret 2 exists
     THEN SFX=176:SGY=168:GOTO 2925       fires shot from turret 2
2900 IF B5X>0 THEN SFX=187:               otherwise, if turret 3
     SGY=170                              exists, fire shot
2925 SDINC=-INC*4                         shot moves up at 4 times ship speed
2950 SGY=SGY+SDINC                        computes next y position of shot
2975 IF SGY<1 THEN SDINC=0:SGY=0:         if shot past top of screen, forget it
     SFX=0
3525 HPLOT LAST%(IDX,6),LAST%(IDX,7)      erases the last position of the shot
3700 LAST%(IDX,6)=SFX:LAST%(IDX,7)=SGY    remembers the last position of the shot
3750 HPLOT SFX,SGY                        plots the current shot from the turret
```



You can probably guess what we're going to do next. Shots were fired, right? Right. We'd better see if anything was hit. Did the shuttle take a hit? Checking this is a little more difficult.

Because of the shape of the shuttle, we can't compare it to a box like we did with the turrets and the base. We're going to need a programming trick to determine if the shuttle has been hit. First we determine if the location where we are going to plot the shot has a part of the ship already plotted on it. The special routines that do this are really too complex to explain here. Like the shape drawing program, you don't have to know exactly how they work to use them in games. Here's the routine you need, type it in.



```
125  GOTO 5425                                          pokes routine into memory
300  IF SFX=0 THEN RETURN                               don't check if there is no shot
350  IF (X-SFX)*(X-SFX)+                                checks if shot is within
     (Y-SGY)*(Y-SGY)>TFS+PEEK(229)                      checking range
     THEN RETURN
400  XPST=SFX:I=-WIDTH                                  calculates memory location
425  XHOFST%=XPST/TFS                                   where shot will be
450  XLOFST%=XPST-TFS*XHOFST%                           drawn
475  I=I+WIDTH:IF I=>-SDINC THEN RETURN
500  YPST=SGY+I
525  IF YPST>168 THEN RETURN
550  CALL 768,YPST,XLOFST%,XHOFST%,62481
575  ADRES=PEEK(38)+PEEK(39)*TFS+PEEK(229)
600  IF PEEK (ADRES)=0 THEN GOTO 475                    all is clear to draw the shot
625  HCOLOR=3:HPLOT SFX,SGY                             plot the shot
650  POKE -16298-WPAGE,0                                go back to previous page
                                                        drawn on
675  SDINC=0:SFX=0:SGY=0                                cancel the shot
725  HCOLOR=3
750  GOTO 1650                                          go do the other CRASH! things
3725  GOSUB 300                                         check to see if shot hit ship
4400  TFS=256
4875  WIDTH=1
5425  A$="300:20 F5 E6 8A 48 20 F5 E6 8A 48 20 F5 E6 8A 48 20
      BE DE 20 67 DD 20 52 E7 68 A8 68 AA 68 6C 50 00 20"
5450  A$=A$+"N D823G"
5475  FOR I=1 TO LEN(A$):POKE 511+I,ASC(MID$(A$,I,1))+128:NEXT :POKE 72,0:CALL -144
5500  GOTO 150
```

Lines 5425, 5450 and 5475 employ a BASIC/machine language subroutine known as the Lam technique. (Luebbert, William F., "What's Where in the Apple" (Micro Ink, Chelmsford, Mass.): p. 38, 39.)

AND THE WINNER IS

This final addition will keep track of the score and announce whether the player or the computer has won the game. If the shuttle shoots all three turrets before getting shot three times, the player wins. If the player gets shot three times before getting all the turrets, the computer wins. Sorry player, try again.

Here's the score keeping code:

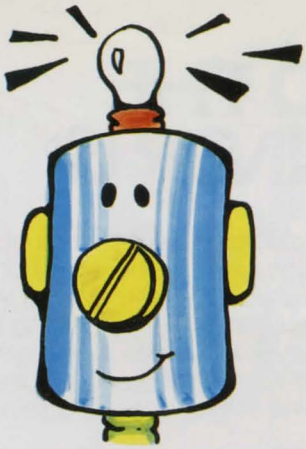
```
4900 TSCRE=0:TBLW=0          TSCRE counts # of turrets hit:
                             TBLW counts # of shuttle hits
250  IF TSCRE=3 OR TBLW=3    if either counter reaches 3
     THEN GOTO 4975          then the game is over
1425 TSCRE = TSCRE + 1       another turret destroyed, add to counter
1800 TBLW = TBLW + 1         shuttle destroyed, add to counter
5000 HOME:VTAB 10:FLASH      sets up to display message
5025 IF TSCRE=3 THEN         if all turrets destroyed
     PRINT "YOU WIN! I GIVE  print congratulations message
     UP!"
5050 IF TBLW = 3 THEN        if shuttle hit 3 times
     PRINT "I WIN, TRY AGAIN" oh well, try again next time
5100 NORMAL:END              NORMAL cancels the FLASH command
```

Save Appleflight.

GETTING EVEN BETTER

Now you know all the techniques required to make your own arcade style game, in fact more. Most games don't need all the techniques we've used. For example, we've shown you two methods of detecting collisions, one by estimating the figure as a box and the other by actually "looking" at the screen to see if we were drawing over something.

The methods and shapes you use can really affect the speed of your game. Notice that the game seems to slow when a shot from the base gets near the shuttle? This is because the shuttle is so thin and the shot travels in such large increments that we have to check for a collision at many points in between shot positions. How could we solve this problem? We could make the shuttle fatter so that WIDTH of the shuttle increased or we could make the program check for a collsion only when the shot was really close to the shuttle.



This process of trying to speed the program up is called "optimization." Through lots of experience, programmers have learned a number of other optimizing methods. Here are a few to keep in mind, but remember, the time to optimize is AFTER you have the program working and you understand it totally. Then you can add the shortcuts that will make your program run faster.

Optimization Tips:

1. Include many statements in one line
   example: 10 X=0:Y=0:GOSUB 300
   instead of:
   ```
   10  X = 0
   20  X =0
   30  GOSUB 300
   ```

2. Avoid GOTO statements
   example: 10 IF X = 0 OR X = 1 THEN Y = 4
   instead of:
   ```
   10  IF X = 0 GOTO 40
   20  IF X = 1 GOTO 40
   30  GOTO 50
   40  Y=4
   50
   ```

3. Remove all REM statements. Only do this once the program works, but watch out for GOTOs and GOSUBs that go to these statements.

4. If a table is going to store numbers that don't have decimals, then always add a % to the table name so it will use up less computer memory. (Keep this trick in mind when you are developing your game and type it in as you go along to avoid lots of retyping later.)
   example: DIM LAST% (8,2)
   instead of DIM LAST (8,2)

5. Don't DRAW anything you don't have to. We can use this rule to optimize the Appleflight program with the following lines of code. Save and run the program. Do you notice a difference and do any new problems crop up? These are the questions you have to answer when optimizing your game.

```
4550  C1RASH = 2                          C1RASH tells us to draw the turrets on graphic pages
1450  C1RASH = 2                          set in whenever there is a collision
1675  C1RASH = 2
3755  IF C1RASH = 0 THEN GOTO 3900        don't plot turrets unless a crash
3775  C1RASH = C1RASH - 1                 so we replot turrets on both pages
3400  IF C1RASH=0 THEN GOTO 3500          don't erase turrets unless a crash
```

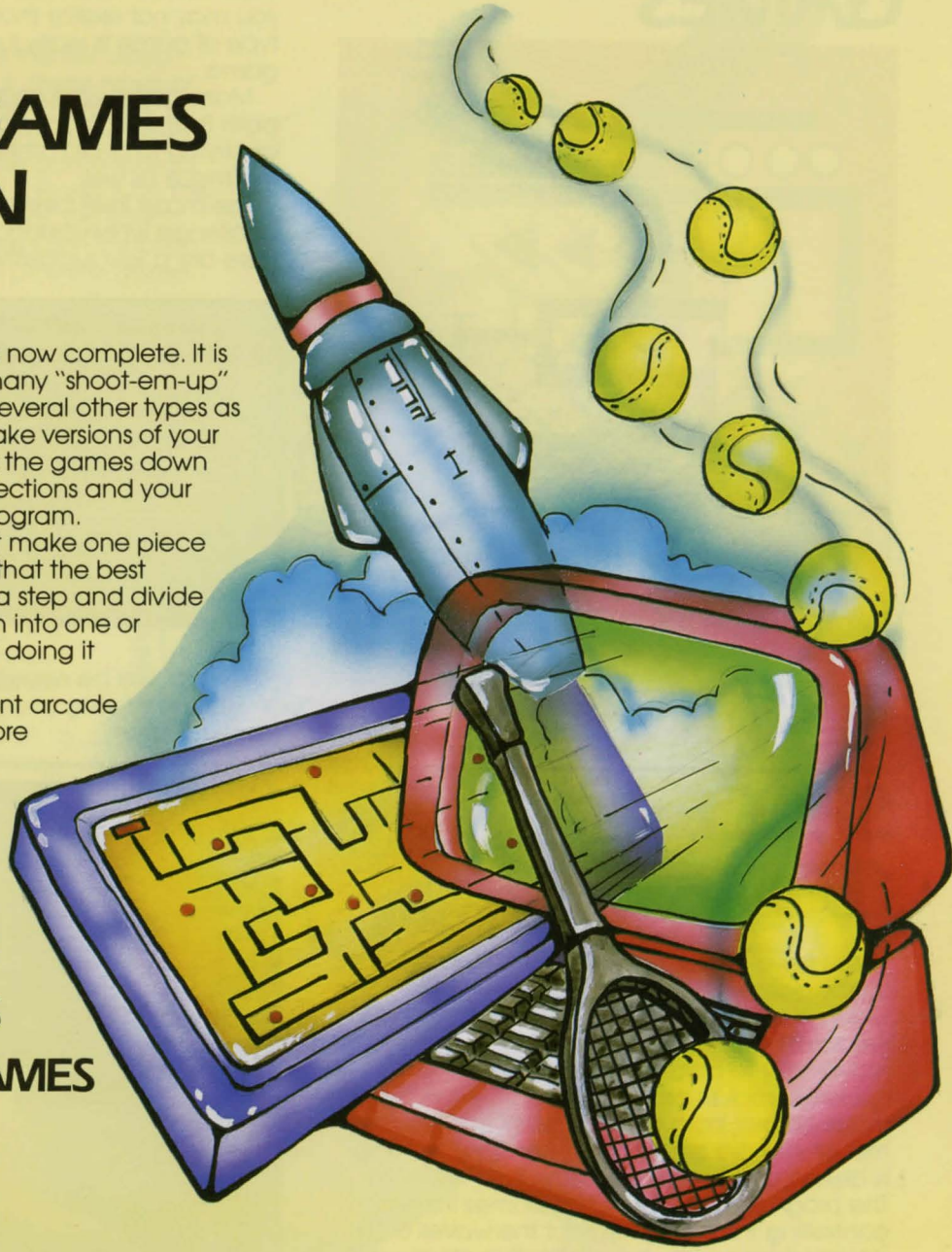
We've already given you all the code you need for the game, but you feel free to optimize it further and maybe even add some things. You may be wondering what use shape four has, the one that looks like a bunch of exploded pieces? Why don't you try to add this shape to the game when the turrets explode?

OTHER GAMES YOU CAN MAKE

The Appleflight game is now complete. It is just one variation of the many "shoot-em-up" space games. There are several other types as well and now you can make versions of your own. Remember to break the games down into small, easy-to-solve sections and your game will be simple to program.

If you get stuck, or can't make one piece work, you'll probably find that the best thing to do is to go back a step and divide the step you're working on into one or two smaller steps, then try doing it again.

There are a lot of different arcade games and more and more keep coming out every day, but the games that seem to be the most popular fall into three categories:



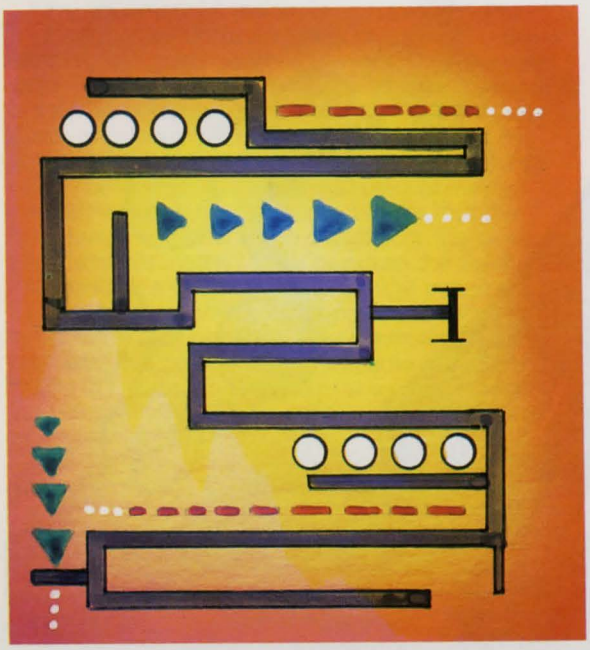
SPACE GAMES
MAZE GAMES
BAT&BALL GAMES

Each type has its own special programming requirements but they can all be programmed by using the techniques you've learned here.

You'll find that you can use many parts of the Appleflight program when developing your own games. The shape table program, for example, will be helpful when you want to create shapes for your games. You should follow the path we took, start small and work up. When you make your own game, you may find that the first few routines are particularly helpful as a starting point.

The next few pages tell a little bit about what characterizes each of these 3 main types of game and special considerations and programming tips for writing your own games of each kind.

MAZE GAMES



Maze games are perhaps the most popular of the arcade games on the market today. Games like Pac Man and Ms Pac Man are the most famous example of this type. However, you may not realize that the Donkey Kong type of game is really just another maze game.

Maze games are characterized by the fixed path that the "man" has to travel. Usually, you are trying to avoid other creatures that travel the maze as well.

The maze itself presents the greatest challenge when creating a maze type game. Here are a few suggestions:

1. Try to keep the maze simple.
2. Do not allow diagonal movement.
3. Concentrate first on making the character travel a fixed path, then add the maze shape to coincide with your path.
4. Get the man moving in the path, only then add other characters in the maze.
5. Draw all your characters so that you won't have to use the ROTate command when they turn corners.

That should be enough hints to get you started at making your own maze games. Have fun!

SPACE GAMES

You have now seen and written one type of space game. The Appleflight game is the type which allows the ship to roam freely around the screen. Other games restrict the movement to a side-to-side motion which generally makes programming much easier. In these games, both collision detection and movement are easier to control since the ship is always drawn with the same y co-ordinate. The programming challenge comes from controlling the movements of the waves of enemy ships which shoot and dive at your ship.

Keep all shapes more box-like than stick-like, this makes collision detection easier.

If you have a large number of enemy ships, use DIM statements to store x, y, old x, old y etc.

Move only a few enemy ships at a time and don't redraw all enemy ships on each move.

BAT & BALL GAMES

Bat and ball games were the first video arcade games to come out. These types of games were relatively simple at first, the object was to use a bat or paddle to keep a ball in an enclosed area. The game spread like wildfire and soon many variations on this same theme began to appear. On your System Master diskette is one such variation, "Little Brick Out." In order to play this game you require game paddles.

This is a low res game but is still a lot of fun. To start playing, type:

RUN LITTLE BRICK OUT

Be sure you have the System Master diskette in Drive 1.

The major problem to solve when writing your own bat and ball games is determining the angle the ball should be deflected when it strikes the wall or bat. This brings to mind complicated mathematics, with many angles and formulas; however, what all that math is really saying is that the ball should be deflected at the same angle at which it approached the wall. To implement this in programming, supposing we struck the top wall, we would simply code:

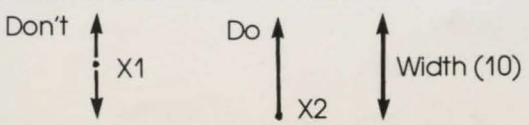
999 YINC = −YINC

(assuming the variable YINC is used to move the ball in the Y direction).

Likewise, if a side wall was struck, we would simply reverse the sign of the XINC variable. (Your teacher should be able to explain the mathematics of why reversing the sign on the INC means that the angle in equals the angle out.)

When checking for collision with the bat don't worry about the thickness of the bat. If the ball has passed the bat's x-co-ordinate and the ball is between the bat's ends, then you have a collision.

Draw or plot the bat shape from end to end rather than from the middle out:



This makes collision checking easier. In the first example you have to check X1 −5 to X to X1 + 5, but in the second you only have to check X2 to X2+10.

GLOSSARY

code –	a language a computer can understand, where characters represent data; a program; (verb) to program
collision detection –	in graphics or games applications, discovering whether one screen drawn image has "collided" with another, can be done by deciding if one image will have to be drawn over another or by PEEKing specific screen locations
co-ordinates –	describe a position or spot on the screen by telling which column and row it is in (X is generally row and y, column)
graphics –	not words or text, but pictures or graphs; of drawing, diagrams or symbolic curves
hi res –	with graphics, good picture quality, drawing with individual pixels for sharp, clear pictures; high pixel density
loop –	section of a program that is designed to perform the same routine (set of instructions) over and over for a specified number of times
lo res –	with graphics, pictures that use blocks of pixels as opposed to high resolution which uses individual pixels, less well defined graphics; low pixel density
machine language –	a very low level language that the computer can understand directly; binary
mainline (program) –	the central controlling part of a program that calls up the subroutines

memory –	space where computer stores information and/or instructions – Read Only Memory is a permanent store of operating instructions which cannot be changed, Random Access Memory is user-programmable and changeable
page flipping –	alternating on the screen, two areas of screen graphics memory, known as pages
PEEK –	BASIC command to examine a memory location
pixel –	short for picture element, each of the tiny lights that make up a character or portion of a picture on a display screen
POKE –	BASIC command to put something into a specific memory location
pseudocode –	simplified English version of the logical procedure for solving a problem
ROTation –	in graphics, the turning or revolving of an image around a fixed point
subroutine –	a small program or routine within a larger one that does one specific thing
variable –	something which represents something else, eg. B=12 – the variable "B" has a value of or represents the number 12

HAYES PUBLISHING LTD.
3312 MAINWAY
BURLINGTON, ONTARIO
L7M 1A7
ISBN 0-88625-047-1